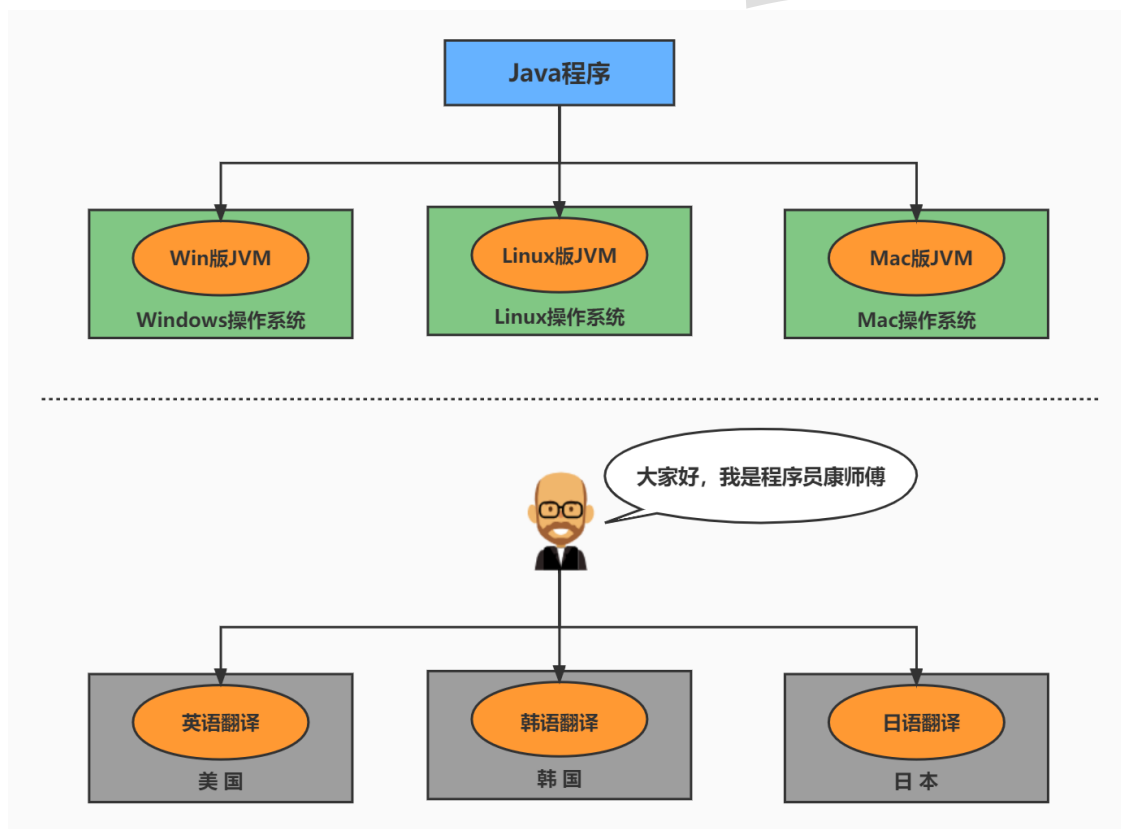


14.1.1 优点

跨平台性：这是 Java 的核心优势。Java 在最初设计时就注重移植和跨平台性。比如：Java 的 int 永远都是 32 位。不像 C++ 可能是 16，32，可能是根据编译器厂商规定的变化。

- 通过 Java 语言编写的应用程序在不同的系统平台上都可以运行。“*Write once , Run Anywhere*”。
- 原理：只要在需要运行 java 应用程序的操作系统上，先安装一个 Java 虚拟机 (JVM , Java Virtual Machine) 即可。由 JVM 来负责 Java 程序在该系统中的运行。



JVM 的跨平台性

Linux	macOS	Windows
不同平台 不同指令集		
Product/file description	File size	Download
Arm 64 Compressed Archive	170.95 MB	https://download.oracle.com/java/17/latest/jdk-17_linux-aarch64_bin.tar.gz (sha256 🔗)
Arm 64 RPM Package	153.12 MB	https://download.oracle.com/java/17/latest/jdk-17_linux-aarch64_bin.rpm (sha256 🔗)
x64 Compressed Archive	172.19 MB	https://download.oracle.com/java/17/latest/jdk-17_linux-x64_bin.tar.gz (sha256 🔗)

面向对象性：

面向对象是一种程序设计技术，非常适合大型软件的设计和开发。面向对象编程支持封装、继承、多态等特性，让程序更好达到高内聚，低耦合的标准。

健壮性：吸收了 C/C++ 语言的优点，但去掉了其影响程序健壮性的部分（如指针、内存的申请与释放等），提供了一个相对安全的内存管理和访问机制。

安全性高：

Java 适合于网络/分布式环境，需要提供一个安全机制以防恶意代码的攻击。如：安全防范机制（ClassLoader 类加载器），可以分配不同的命名空间以防替代本地的同名类、字节代码检查。

简单性：

Java 就是 C++ 语法的简化版，我们也可以将 Java 称之为“C++-”。比如：头文件，指针运算，结构，联合，操作符重载，虚基类等。

高性能：

- Java 最初发展阶段，总是被人诟病“性能低”；客观上，高级语言运行效率总是低于低级语言的，这个无法避免。Java 语言本身发展中通过虚拟机的优化提升了几十倍运行效率。比如，通过 JIT (JUST IN TIME) 即时编译技术提高运行效率。
- Java 低性能的短腿，已经被完全解决了。业界发展上，我们也看到很多 C++ 应用转到 Java 开发，很多 C++ 程序员转型为 Java 程序员。

14.1.2 缺点

- 语法过于复杂、严谨，对程序员的约束比较多，与 python、php 等相比入门较难。但是一旦学会了，就业岗位需求量大，而且薪资待遇节节攀升。
- 一般适用于大型网站开发，整个架构会比较重，对于初创公司开发和维护人员的成本比较高（即薪资高），选择用 Java 语言开发网站或应用系统的需要一定的经济实力。

- 并非适用于所有领域。比如，Objective C、Swift 在 iOS 设备上就有着无可取代的地位。浏览器中的处理几乎完全由 JavaScript 掌控。Windows 程序通常都用 C++ 或 C# 编写。Java 在服务器端编程和跨平台客户端应用领域则很有优势。

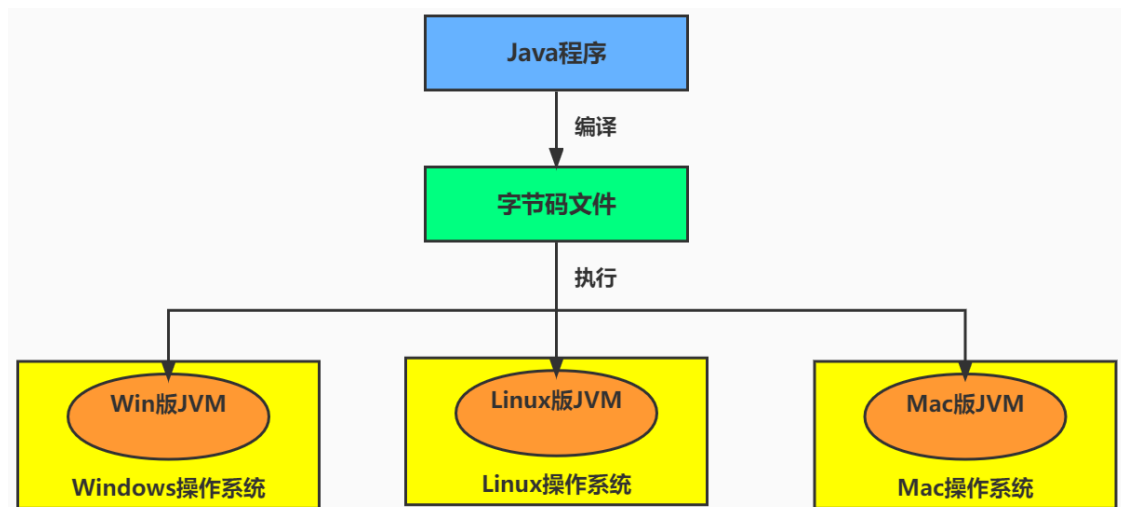
14.2 JVM 功能说明

JVM (Java Virtual Machine , Java 虚拟机)：是一个虚拟的计算机，是 Java 程序的运行环境。JVM 具有指令集并使用不同的存储区域，负责执行指令，管理数据、内存、寄存器。

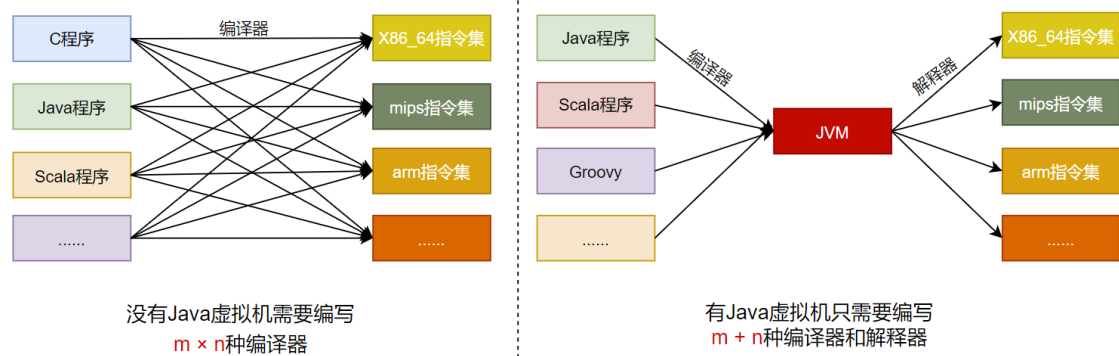


14.2.1 功能 1：实现 Java 程序的跨平台性

我们编写的 Java 代码，都运行在 **JVM** 之上。正是因为有了 JVM，才使得 Java 程序具备了跨平台性。



使用 JVM 前后对比：



14.2.2 功能 2：自动内存管理(内存分配、内存回收)

- Java 程序在运行过程中，涉及到运算的**数据的分配、存储**等都由 JVM 来完成
- Java 消除了程序员回收无用内存空间的职责。提供了一种系统级线程跟踪存储空间的分配情况，在内存空间达到相应阈值时，检查并释放可被释放的存储器空间。
- GC 的自动回收，提高了内存空间的利用效率，也提高了编程人员的效率，很大程度上**减少了**因为没有释放空间而导致的**内存泄漏**。

面试题：

Java 程序还会出现内存溢出和内存泄漏问题吗？ Yes!

15. 章节案例

案例 1：个人信息输出

```
姓名：康师傅
性别：男
家庭住址：北京程序员聚集地：回龙观
```

```
class Exercise1{
    public static void main(String[] args){
        System.out.println("姓名：康师傅");
        System.out.println();//换行操作
        System.out.println("性别：男");
        System.out.println("家庭住址：北京程序员聚集地：回龙观");
    }
}
```

案例 2：输出：心形

结合\n(换行), \t(制表符), 空格等在控制台打印出如下图所示的效果。

```

      *
    *   *
  *       *
 *         *
*           *
 *         *
  *       *
    *   *
      *

      I love Java

      *
    *   *
  *       *
 *         *
*           *
 *         *
  *       *
    *   *
      *
```

方式一：

//方式一:

```
class Exercise2{
    public static void main(String[] args){
        System.out.print("\t");
        System.out.print("*");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");

        System.out.println("*");

        System.out.print("*");
        System.out.print("\t");
        //System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("I love java");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");
        System.out.println("*");

        System.out.print("\t");
        System.out.print("*");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");
        System.out.print("\t");

        System.out.println("*");

        System.out.print("\t");
```

```
System.out.print("\t");
System.out.print("*");
System.out.print("\t");
System.out.print("\t");
System.out.print("\t");
System.out.print("\t");
System.out.print("\t");

System.out.println("*");
```

```
System.out.print("\t");
System.out.print("\t");
System.out.print("\t");
System.out.print("*");
System.out.print("\t");
System.out.print("\t");
System.out.print("\t");

System.out.println("*");
```

```
System.out.print("\t");
System.out.print("\t");
System.out.print("\t");
System.out.print("\t");
System.out.print("*");
System.out.print("\t");

System.out.println("*");
```

```
System.out.print("\t");
System.out.print("\t");
System.out.print("\t");
System.out.print("\t");
System.out.print("  ");
System.out.print("*");
```

```
}
```

```
}
```

方式二：

第 02 章_变量与运算符

本章专题与脉络

第02章-变量与运算符

专题2-1: 关键字、保留字与标识符

关键字与保留字

● 标识符命名规则、规范

专题2-2: 变量的分类与定义、注意点

● 变量按数据类型的分类

● 变量的定义格式

专题2-3: 整型和浮点型变量的使用

整型: byte、short、int、long

浮点型: float、double

● 案例: 根据圆周率求面积

● 案例: 华氏温度转摄氏温度

专题2-4: 字符型和布尔型变量的使用

★ 字符型: char

布尔型: boolean

专题2-5: 基本数据类型变量的自动类型提升

● 提升规则

★ 特别的: byte、short、char

专题2-6: 基本数据类型变量的强制类型转换

● 转换格式

精度损失问题

专题2-7: String与基本数据类型变量间的运算

连接符: +

运算规则

● 案例: 公安局身份登记

专题2-8: 二进制的表示、二进制与十进制间的转换

★ 原码、反码、补码

二进制->十进制: 权相加法

十进制->二进制: 除2取余的逆

专题2-9: 二进制与八进制、十六进制间的转换

各进制间的转换

专题2-10: 算术运算符

★ ++、--

● 案例: 获取三位数各个位的值

● 案例: 计算抗洪时间

专题2-11: 赋值运算符

● 案例: 你有几种办法实现变量加1、加2

专题2-12: 比较运算符、逻辑运算符

★ = 与 ==

★ & 与 &&、| 与 ||

● 面试: 结果判断

● 案例: 赋值操作

★ 专题2-13: 位运算符

● 案例: 变量换值

● 案例: 高效计算2*8

专题2-14: 条件运算符与运算符的优先级

● 案例: 求两个数、三个数的最大值

● 案例: 10天后是周几

● 专题2-15: 本章随堂复习与企业真题

第 1 阶段：Java 基本语法-第 02 章

1. 关键字 (keyword)

定义：被 Java 语言赋予了特殊含义，用做专门用途的字符串（或单词） HelloWorld 案例中，出现的关键字有 `class`、`public`、`static`、`void` 等，这些单词已经被 Java 定义好了。

特点：全部关键字都是小写字母。

关键字比较多，不需要死记硬背，学到哪里记到哪里即可。

官方地址：https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

Java Language Keywords

Here is a list of keywords in the Java programming language. You cannot use any of the following as identifiers in your programs. The keywords `const` and `goto` are reserved, even though they are not currently used. `true`, `false`, and `null` might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

<code>abstract</code>	<code>continue</code>	<code>for</code>	<code>new</code>	<code>switch</code>
<code>assert</code> ***	<code>default</code>	<code>goto</code> *	<code>package</code>	<code>synchronized</code>
<code>boolean</code>	<code>do</code>	<code>if</code>	<code>private</code>	<code>this</code>
<code>break</code>	<code>double</code>	<code>implements</code>	<code>protected</code>	<code>throw</code>
<code>byte</code>	<code>else</code>	<code>import</code>	<code>public</code>	<code>throws</code>
<code>case</code>	<code>enum</code> ****	<code>instanceof</code>	<code>return</code>	<code>transient</code>
<code>catch</code>	<code>extends</code>	<code>int</code>	<code>short</code>	<code>try</code>
<code>char</code>	<code>final</code>	<code>interface</code>	<code>static</code>	<code>void</code>
<code>class</code>	<code>finally</code>	<code>long</code>	<code>strictfp</code> **	<code>volatile</code>
<code>const</code> *	<code>float</code>	<code>native</code>	<code>super</code>	<code>while</code>
* not used				
** added in 1.2				
*** added in 1.4				
**** added in 5.0				

说明：

- 关键字一共 50 个，其中 `const` 和 `goto` 是保留字(reserved word)。
- `true`、`false`、`null` 不在其中，它们看起来像关键字，其实是字面量，表示特殊的布尔值和空值。

用于定义数据类型的关键字				
class	interface	enum	byte	short
int	long	float	double	char
boolean	void			
用于定义流程控制的关键字				
if	else	switch	case	default
while	do	for	break	continue
return				
用于定义访问权限修饰符的关键字				
private	protected	public		
用于定义类，函数，变量修饰符的关键字				
abstract	final	static	synchronized	
用于定义类与类之间关系的关键字				
extends	implements			
用于定义建立实例及引用实例，判断实例的关键字				
new	this	super	instanceof	
用于异常处理的关键字				
try	catch	finally	throw	throws
用于包的关键字				
package	import			
其他修饰符关键字				
native	strictfp	transient	volatile	assert
const	goto			
* 用于定义数据类型值的字面值				
true	false	null		

2. 标识符(identifier)

Java 中变量、方法、类等要素命名时使用的字符序列，称为标识符。

技巧：凡是自己可以起名字的地方都叫标识符。

标识符的命名规则（必须遵守的硬性规定）：

- 由 26 个英文字母大小写，0-9，_或 \$ 组成
- 数字不可以开头。
- 不可以使用关键字和保留字，但能包含关键字和保留字。
- Java 中严格区分大小写，长度无限制。
- 标识符不能包含空格。

练习：miles、Test、a++、--a、4#R、\$4、#44、apps、class、
public、int、x、y、radius

标识符的命名规范（建议遵守的软性要求，否则工作时容易被鄙视）：

- ＞ 包名：多单词组成时所有字母都小写：xxxyyyzzz。
例如：java.lang、com.atguigu.bean
- ＞ 类名、接口名：多单词组成时，所有单词的首字母大写：XxxYyyZzz
例如：HelloWorld、String、System 等
- ＞ 变量名、方法名：多单词组成时，第一个单词首字母小写，第二个单词开始每个单词首字母大写：xxxYyyZzz
例如：age、name、bookName、main、binarySearch、getName
- ＞ 常量名：所有字母都大写。多单词时每个单词用下划线连接：XXX_YYY_ZZZ
例如：MAX_VALUE、PI、DEFAULT_CAPACITY

注意：在起名字时，为了提高阅读性，要尽量有意义，“见名知意”。

更多细节详见《代码整洁之道_关于标识符.txt》《阿里巴巴 Java 开发手册-1.7.1-黄山版》

3. 变量

3.1 为什么需要变量

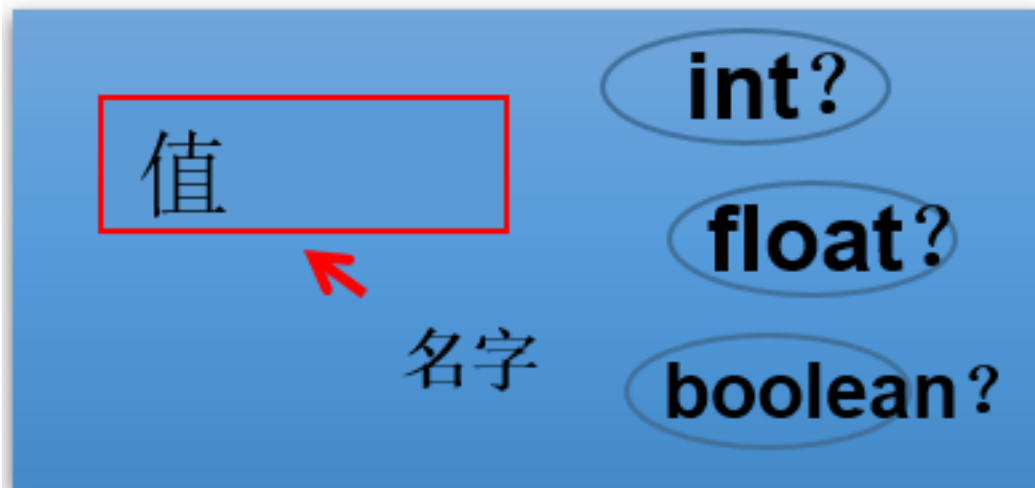


一花一世界，如果把一个程序看做一个世界或一个社会的话，那么变量就是程序世界的花草草、万事万物。即，变量是程序中不可或缺的组成单位，最基本的存储单元。



3.2 初识变量

- 变量的概念：
 - 内存中的一个存储区域，该区域的数据可以在同一类型范围内不断变化
 - 变量的构成包含三个要素：数据类型、变量名、存储的值
 - Java 中变量声明的格式：数据类型 变量名 = 变量值

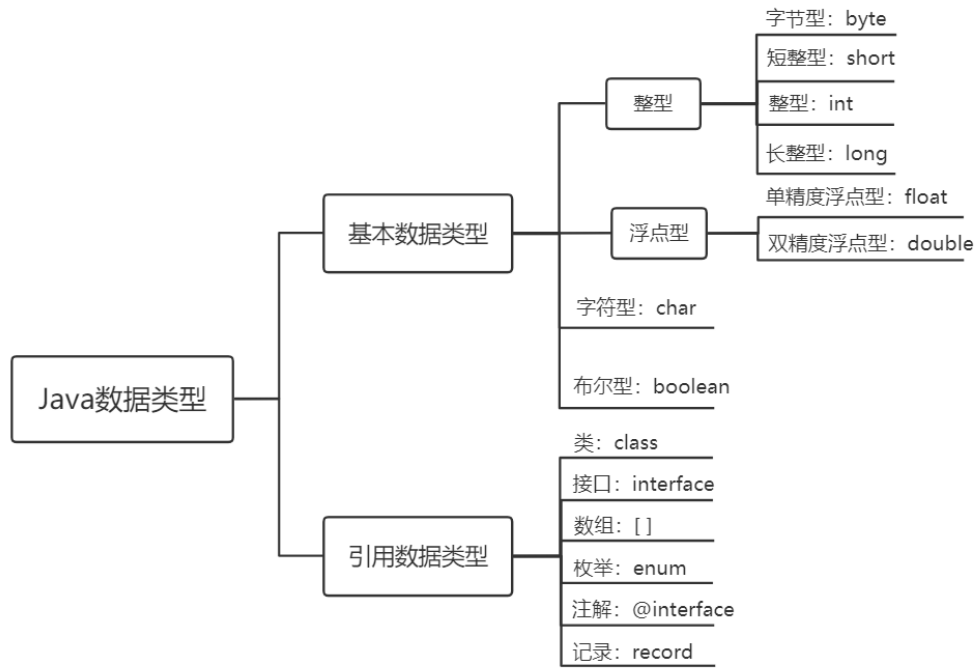


- 变量的作用：用于在内存中保存数据。
- 使用变量注意：
 - Java 中每个变量必须先声明，后使用。
 - 使用变量名来访问这块区域的数据。
 - 变量的作用域：其定义所在的一对{ }内。
 - 变量只有在其 *作用域*内才有效。出了作用域，变量不可以再被调用。
 - 同一个作用域内，不能定义重名的变量。

3.3 Java 中变量的数据类型

Java 中变量的数据类型分为两大类：

- **基本数据类型**：包括 *整数类型*、*浮点数类型*、*字符类型*、*布尔类型*。
- **引用数据类型**：包括 *数组*、*类*、*接口*、*枚举*、*注解*、*记录*。



3.4 变量的使用

3.4.1 步骤 1: 变量的声明

格式: 数据类型 变量名;

// 例如:

// 存储一个整数类型的年龄

```
int age;
```

// 存储一个小数类型的体重

```
double weight;
```

// 存储一个单字符类型的性别

```
char gender;
```

// 存储一个布尔类型的婚姻状态

```
boolean marry;
```

// 存储一个字符串类型的姓名

```
String name;
```

// 声明多个同类型的变量

```
int a,b,c; // 表示 a,b,c 三个变量都是 int 类型。
```

注意：变量的数据类型可以是基本数据类型，也可以是引用数据类型。

3.4.2 步骤 2：变量的赋值

给变量赋值，就是把“值”存到该变量代表的内存空间中。同时，给变量赋的值类型必须与变量声明的类型一致或兼容。

变量赋值的语法格式：

变量名 = 值；

举例 1：可以使用合适类型的 *常量值* 给已经声明的变量赋值

```
age = 18;  
weight = 109;  
gender = '女';
```

举例 2：可以使用其他 *变量* 或者 *表达式* 给变量赋值

```
int m = 1;  
int n = m;  
  
int x = 1;  
int y = 2;  
int z = 2 * x + y;
```

变量可以反复赋值

//先声明，后初始化

```
char gender;  
gender = '女';
```

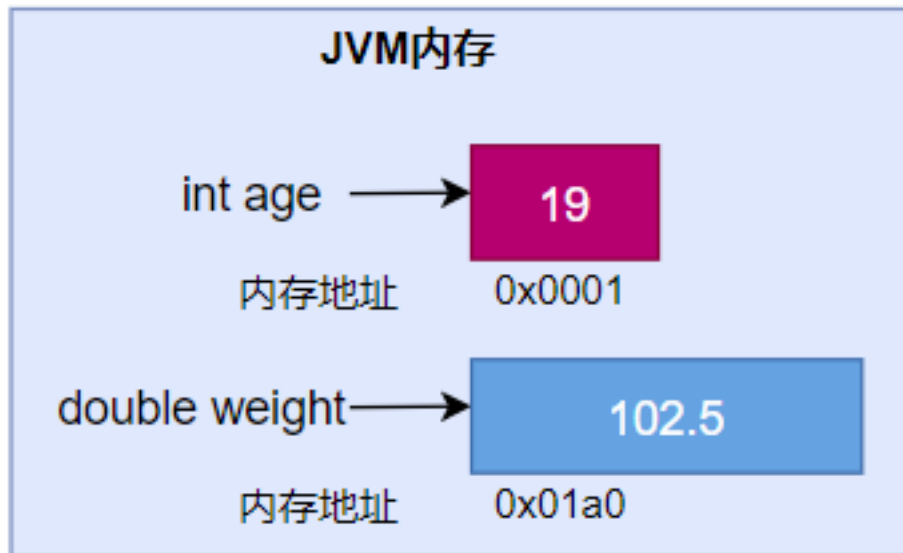
//给变量重新赋值，修改 gender 变量的值

```
gender = '男';  
System.out.println("gender = " + gender);//gender = 男
```

举例 4：也可以将变量的声明和赋值一并执行


```
boolean isBeauty = true;  
String name = "迪丽热巴";
```

内存结构如图：



4. 基本数据类型介绍

4.1 整数类型：byte、short、int、long



Java 各整数类型有固定的表数范围和字段长度，不受具体操作系统的影响，以保证 Java 程序的可移植性。

类 型	占用存储空间	表数范围
byte	1字节=8bit位	-128 ~ 127
short	2字节	$-2^{15} \sim 2^{15}-1$
int	4字节	$-2^{31} \sim 2^{31}-1$ (约21亿)
long	8字节	$-2^{63} \sim 2^{63}-1$

- 定义 long 类型的变量，赋值时需要以 "L" 或 "l" 作为后缀。
- Java 程序中变量通常声明为 int 型，除非不足以表示较大的数，才使用 long。
- Java 的整型常量默认为 int 型。

4.1.1 补充：计算机存储单位

字节 (Byte)：是计算机用于计量存储容量的基本单位，一个字节等于 8 bit。

位 (bit)：是数据存储的最小单位。二进制数系统中，每个 0 或 1 就是一个位，叫做 bit (比特)，其中 8 bit 就称为一个字节(Byte)。

转换关系：

- 8 bit = 1 Byte
- 1024 Byte = 1 KB
- 1024 KB = 1 MB
- 1024 MB = 1 GB
- 1024 GB = 1 TB

4.2 浮点类型：float、double

与整数类型类似，Java 浮点类型也有固定的表数范围和字段长度，不受具体操作系统的影响。

类 型	占用存储空间	表数范围
单精度float	4字节	$-3.403E38 \sim 3.403E38$
双精度double	8字节	$-1.798E308 \sim 1.798E308$

浮点型常量有两种表示形式：

- 十进制数形式。如：5.12 512.0f .512 (必须有小数点)
- 科学计数法形式。如：5.12e2 512E2 100E-2

float：单精度，尾数可以精确到 7 位有效数字。很多情况下，精度很难满足需求。

double：双精度，精度是 float 的两倍。通常采用此类型。

定义 float 类型的变量，赋值时需要以"f"或"F"作为后缀。

Java 的浮点型常量默认为 double 型。

4.2.1 关于浮点型精度的说明

- 并不是所有的小数都能可以精确的用二进制浮点数表示。二进制浮点数不能精确的表示 0.1、0.01、0.001 这样 10 的负次幂。
- 浮点类型 float、double 的数据不适合在不容许舍入误差的金融计算领域。如果需要精确数字计算或保留指定位数的精度，需要使用 *BigDecimal* 类。

测试用例：

```
//测试1：（解释见章末企业真题：为什么 0.1 + 0.2 不等于 0.3）
System.out.println(0.1 + 0.2); //0.30000000000000004
```

```
//测试2：
float ff1 = 123123123f;
float ff2 = ff1 + 1;
System.out.println(ff1);
System.out.println(ff2);
System.out.println(ff1 == ff2);
```

4.2.2 应用举例

案例 1：定义圆周率并赋值为 3.14，现有 3 个圆的半径分别为 1.2、2.5、6，求它们的面积。

```
/**
 * @author 尚硅谷-宋红康
 * @create 12:36
 */
public class Exercise1 {
```

```

public static void main(String[] args) {
    double PI = 3.14; //圆周率

    double radius1 = 1.2;
    double radius2 = 2.5;
    int radius3 = 6;

    System.out.println("第 1 个圆的面积: " + PI * radius1 * radius
1);
    System.out.println("第 2 个圆的面积: " + PI * radius2 * radius
2);
    System.out.println("第 3 个圆的面积: " + PI * radius3 * radius
3);
}
}

```

案例 2：小明要到美国旅游，可是那里的温度是以华氏度为单位记录的。它需要一个程序将华氏温度（80 度）转换为摄氏度，并以华氏度和摄氏度为单位分别显示该温度。

$$^{\circ}\text{C} = (^{\circ}\text{F} - 32) / 1.8$$

```

/**
 * @author 尚硅谷-宋红康
 * @create 12:51
 */
public class Exercise2 {
    public static void main(String[] args) {
        double hua = 80;
        double she = (hua-32)/1.8;
        System.out.println("华氏度" + hua+"°F转为摄氏度是" +she+"°C");
    }
}

```

4.3 字符类型：char

char 型数据用来表示通常意义上“字符”（占 2 字节）

Java 中的所有字符都使用 Unicode 编码，故一个字符可以存储一个字母，一个汉字，或其他书面语的一个字符。

字符型变量的三种表现形式：

- **形式 1:** 使用单引号(' ')括起来的单个字符。

例如: char c1 = 'a'; char c2 = '中'; char c3 = '9';

- **形式 2:** 直接使用 *Unicode* 值来表示字符型常量: '\uXXXX'。其中, XXXX 代表一个十六进制整数。

例如: \u0023 表示 '#'。

- **形式 3:** Java 中还允许使用转义字符'\ '来将其后的字符转变为特殊字符型常量。

例如: char c3 = '\n'; // '\n'表示换行符

转义字符	说明	Unicode 表示方式
\n	换行符	\u000a
\t	制表符	\u0009
\"	双引号	\u0022
\'	单引号	\u0027
\\	反斜线	\u005c
\b	退格符	\u0008
\r	回车符	\u000d

char 类型是可以进行运算的。因为它都对应应有 Unicode 码, 可以看做是一个数值。

4.4 布尔类型：boolean

boolean 类型用来判断逻辑条件，一般用于流程控制语句中：

- if 条件控制语句；
- while 循环控制语句；
- for 循环控制语句；
- do-while 循环控制语句；

boolean 类型数据只有两个值：true、false，无其它

- 不可以使用 0 或非 0 的整数替代 false 和 true，这点和 C 语言不同。
- 拓展：Java 虚拟机中没有任何供 boolean 值专用的字节码指令，Java 语言表达所操作的 boolean 值，在编译之后都使用 java 虚拟机中的 int 数据类型来代替：true 用 1 表示，false 用 0 表示。——《java 虚拟机规范 8 版》

举例：

```
boolean isFlag = true;

if(isFlag){
    //true 分支
}else{
    //false 分支
}
```

经验之谈：

Less is More! 建议不要这样写：if (isFlag == true)，只有新手才如此。关键也很容易写错成 if(isFlag = true)，这样就变成赋值 isFlag 为 true 而不是判断！老鸟的写法是 if (isFlag)或者 if (!isFlag)。

5. 基本数据类型变量间运算规则

在 Java 程序中，不同的基本数据类型（只有 7 种，不包含 boolean 类型）变量的值经常需要进行相互转换。

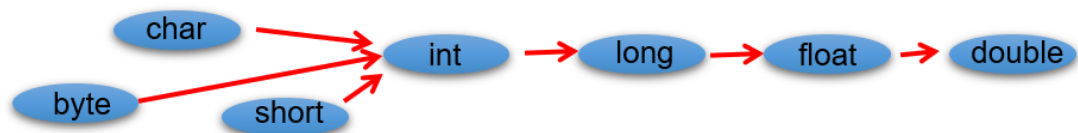
转换的方式有两种：自动类型提升和强制类型转换。

5.1 自动类型提升

规则：将取值范围小（或容量小）的类型自动提升为取值范围大（或容量大）的类型。



基本数据类型的转换规则如图所示：



(1) 当把存储范围小的值（常量值、变量的值、表达式计算的结果值）赋值给了存储范围大的变量时

```
int i = 'A'; // char 自动升级为 int， 其实就是把字符的编码值赋值给 i 变量了
double d = 10; // int 自动升级为 double
long num = 1234567; // 右边的整数常量值如果在 int 范围呢，编译和运行都可以通过，这里涉及到数据类型转换
```

```
// byte bigB = 130; // 错误，右边的整数常量值超过 byte 范围
long bigNum = 12345678912L; // 右边的整数常量值如果超过 int 范围，必须加 L，显式表示 Long 类型。否则编译不通过
```

(2) 当存储范围小的数据类型与存储范围大的数据类型变量一起混合运算时，会按照其中最大的类型运算。

```
int i = 1;
byte b = 1;
double d = 1.0;

double sum = i + b + d; //混合运算, 升级为 double
```

(3) 当 byte, short, char 数据类型的变量进行算术运算时，按照 int 类型处理。

```
byte b1 = 1;
byte b2 = 2;
byte b3 = b1 + b2; //编译报错, b1 + b2 自动升级为 int

char c1 = '0';
char c2 = 'A';
int i = c1 + c2; //至少需要使用 int 类型来接收
System.out.println(c1 + c2); //113
```

练习：

设 x 为 float 型变量，y 为 double 型变量，a 为 int 型变量，b 为 long 型变量，c 为 char 型变量，则表达式 $x + y * a / x + b / y + c$ 的值类型为：

A. int B. long C. double D. char

5.2 强制类型转换

将 3.14 赋值到 int 类型变量会发生什么？产生编译失败，肯定无法赋值。

```
int i = 3.14; // 编译报错
```

想要赋值成功，只有通过强制类型转换，将 double 类型强制转换成 int 类型才能赋值。



大房换小房



规则：将取值范围大（或容量大）的类型强制转换成取值范围小（或容量小）的类型。

自动类型提升是 Java 自动执行的，而强制类型转换是自动类型提升的逆运算，需要我们自己手动执行。

转换格式：

数据类型 1 变量名 = (数据类型 1)被强转数据值; *//() 中的数据类型必须<= 变量值的数据类型*

(1) 当把存储范围大的值（常量值、变量的值、表达式计算的结果值）强制转换为存储范围小的变量时，可能会损失精度或溢出。

```
int i = (int)3.14; // 损失精度
```

```
double d = 1.2;  
int num = (int)d; // 损失精度
```

```
int i = 200;  
byte b = (byte)i; // 溢出
```

(2) 当某个值想要提升数据类型时，也可以使用强制类型转换。这种情况的强制类型转换是没有风险的，通常省略。

```
int i = 1;
int j = 2;
double bigger = (double)(i/j);
```

(3) 声明 long 类型变量时，可以出现省略后缀的情况。float 则不同。

```
long l1 = 123L;
long l2 = 123; //如何理解呢？此时可以看做是 int 类型的 123 自动类型提升为 long 类型
```

```
//long l3 = 123123123123; //报错，因为 123123123123 超出了 int 的范围。
long l4 = 123123123123L;
```

```
//float f1 = 12.3; //报错，因为 12.3 看做是 double，不能自动转换为 float 类型
```

```
float f2 = 12.3F;
float f3 = (float)12.3;
```

练习：判断是否能通过编译

```
1) short s = 5;
   s = s-2; //判断: no
2) byte b = 3;
   b = b + 4; //判断: no
   b = (byte)(b+4); //判断: yes
3) char c = 'a';
   int i = 5;
   float d = .314F;
   double result = c+i+d; //判断: yes
4) byte b = 5;
   short s = 3;
   short t = s + b; //判断: no
```

问答：为什么标识符的声明规则里要求不能数字开头？

//如果允许数字开头，则如下的声明编译就可以通过：

```
int 123L = 12;
```

//进而，如下的声明中 L 的值到底是 123？还是变量 123L 对应的取值 12 呢？出现

歧义了。

```
long l = 123L;
```

5.3 基本数据类型与 String 的运算

5.3.1 字符串类型：String

String 不是基本数据类型，属于引用数据类型

使用一对“”来表示一个字符串，内部可以包含 0 个、1 个或多个字符。

声明方式与基本数据类型类似。例如：String str = “尚硅谷”；

5.3.2 运算规则

- 1、任意八种基本数据类型的数据与 String 类型只能进行连接“+”运算，且结果一定也是 String 类型

```
System.out.println("" + 1 + 2); //12
```

```
int num = 10;  
boolean b1 = true;  
String s1 = "abc";
```

```
String s2 = s1 + num + b1;  
System.out.println(s2); //abc10true
```

```
//String s3 = num + b1 + s1; //编译不通过，因为 int 类型不能与 boolean 运算  
String s4 = num + (b1 + s1); //编译通过
```

- 2、String 类型不能通过强制类型()转换，转为其他的类型

```
String str = "123";  
int num = (int)str; //错误的
```

```
int num = Integer.parseInt(str); //正确的，后面才能讲到，借助包装类的方法才能转
```

5.3.3 案例与练习

案例：公安局身份登记

要求填写自己的姓名、年龄、性别、体重、婚姻状况（已婚用 true 表示，单身用 false 表示）、联系方式等等。

```
/**
 * @author 尚硅谷-宋红康
 * @create 12:34
 */
public class Info {
    public static void main(String[] args) {
        String name = "康师傅";
        int age = 37;
        char gender = '男';
        double weight = 145.6;
        boolean isMarried = true;
        String phoneNumber = "13112341234";

        System.out.println("姓名: " + name);
        System.out.println("年龄: " + age);
        System.out.println("性别: " + gender);
        System.out.println("体重: " + weight);
        System.out.println("婚否: " + isMarried);
        System.out.println("电话: " + phoneNumber);
        //或者
        System.out.println("name = " + name + ",age = " + age + ", gender = " +
            gender + ",weight = " + weight + ",isMarried
            = " + isMarried + ",phoneNumber = " + phoneNumber);
    }
}
```

练习：

练习 1:

```
String str1 = 4; //判断对错:
String str2 = 3.5f + ""; //判断str2 对错:
System.out.println(str2); //输出:
System.out.println(3+4+"Hello!"); //输出:
System.out.println("Hello!"+3+4); //输出:
System.out.println('a'+1+"Hello!"); //输出:
System.out.println("Hello"+'a'+1); //输出:
```

练习 2:

```
System.out.println("*    *"); //输出:
System.out.println("*\t*"); //输出:
System.out.println("*" + "\t" + "*"); //输出:
System.out.println('*' + "\t" + "*"); //输出:
System.out.println('*' + '\t' + "*"); //输出:
System.out.println('*' + "\t" + '*'); //输出:
System.out.println("*" + '\t' + '*'); //输出:
System.out.println('*' + '\t' + '*'); //输出:
```

6. 计算机底层如何存储数据

计算机世界中只有二进制，所以计算机中存储和运算的所有数据都要转为二进制。包括数字、字符、图片、声音、视频等。

```
011111111111100111000111111000111010101111100000001100
101111111111111111111011111110011111101110011100000011
0111111110011110011101110010001111111001100110001111110
0100000000101110011011011111101101101110011101111001000
11110111101111101110000100111110111100100111111111011
110011110100001111110000010001000101100111010111111110
1011101011001101111111111110110010110000010101111110011
01111001100011001110011101100001110001111100001110
0110011111111111010001011011100011111110101101100111101
010000001111111111000111111110111111101111111111011111
00100011110111110110001111111001111111011101111100111
101110101101000001110000011011111011001000001101110100
01111111110001110111110111000011101000001000000000011000
11011111100110001111110110101111010111010000111110011001
0111111100111100111111111011011111110001100011100001100
1110001111011111011011110111000110011100010001010110001
0000100111010111000011101111100011010011111111100111011
1010001011110111100001111111001011101001001001000011001
000100101111111100000101001111110101101101111111111101
1111100011110010001110100011011101000111111001110000001
```

010101

世界上有 10 种人，认识和不认识二进制的。

6.1 进制的分类

十进制 (decimal)

- 数字组成: 0-9
- 进位规则: 满十进一

二进制 (binary)

```
class BinaryTest {  
    public static void main(String[] args) {  
  
        int num1 = 123;           //十进制  
        int num2 = 0b101;        //二进制  
        int num3 = 0127;         //八进制  
        int num4 = 0x12aF;       //十六进制  
  
        System.out.println(num1);  
        System.out.println(num2);  
        System.out.println(num3);  
        System.out.println(num4);  
    }  
}
```

6.2 进制的换算举例

十进制	二进制	八进制	十六进制
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5

十进制	二进制	八进制	十六进制
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	a 或 A
11	1011	13	b 或 B
12	1100	14	c 或 C
13	1101	15	d 或 D
14	1110	16	e 或 E
15	1111	17	f 或 F
16	10000	20	10

6.3 二进制的由来

二进制，是计算技术中广泛采用的一种数制，由德国数理哲学大师莱布尼茨于1679年发明。

二进制数据是用0和1两个数码来表示的数。它的基数为2，进位规则是“逢二进一”。

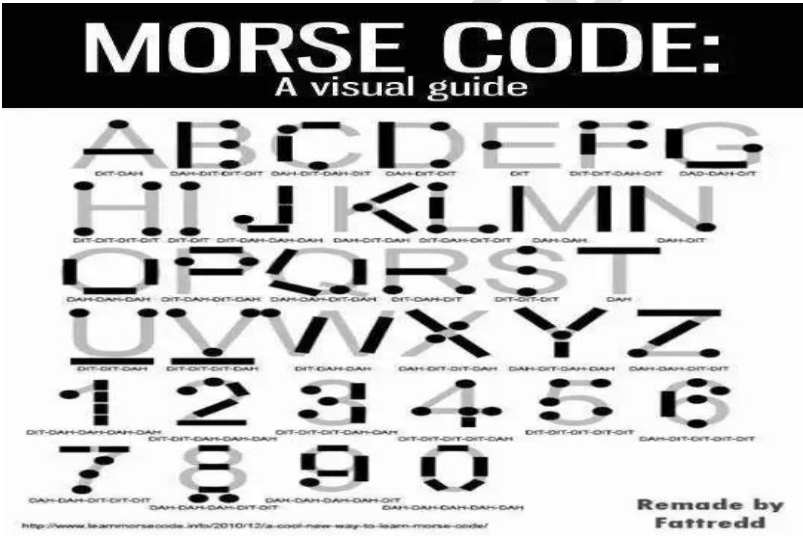
二进制广泛应用于我们生活的方方面面。比如，广泛使用的摩尔斯电码

(Morse Code)，它由两种基本信号组成：短促的点信号“·”，读“滴”；保持一

定时间的长信号“—”，读“嗒”。然后，组成了 26 个字母，从而拼写出相应的单词。

Morse Code			
A	•—	M	—•—
B	—•••	N	—•
C	—••—	O	—•—
D	—••	P	•—••
E	•	Q	—••—
F	••••	R	•—•
G	—•—	S	•••
H	••••	T	—
I	••	U	••—
J	•—•—	V	•••—
K	—••	W	•—•—
L	•—••	X	—••—
		Y	—••—
		Z	—•••
		Ä	•—••
		Ö	—•••
		Ü	••••
		Ch	—••—
		0	—••—
		1	•—••
		2	••—•
		3	••••
		4	••••
		5	••••
		6	—•••
		7	—•••
		8	—•••
		9	—•••
		.	••••
		,	—•••
		?	••••
		!	••••
		:	—•••
		"	••••
		'	—•••
		=	—•••

记忆技巧：



morsecode

我们偶尔会看到的：SOS，即为：



6.4 二进制转十进制

二进制如何表示整数？

计算机数据的存储使用二进制补码形式存储，并且最高位是符号位。

- 正数：最高位是 0
- 负数：最高位是 1

规定

- 正数的补码与反码、原码一样，称为三码合一
- 负数的补码与反码、原码不一样：

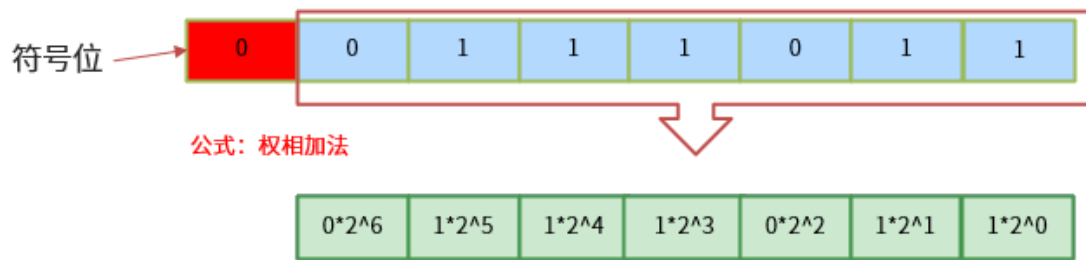
负数的原码：把十进制转为二进制，然后最高位设置为 1

负数的反码：在原码的基础上，最高位不变，其余位取反（0 变 1, 1 变 0）

负数的补码：反码+1

二进制转十进制：权相加法

针对于 byte 数据举例来说：



上述每个框中计算的结果再相加：0 + 32 + 16 + 8 + 2 + 1 = 59

- 例如：byte 类型（1 个字节，8 位）

25 ==> 原码 0001 1001 ==> 反码 0001 1001 --> 补码 0001 1001

-25 ==> 原码 1001 1001 ==> 反码 1110 0110 ==> 补码 1110 0111

整数：

正数：25 00000000 00000000 00000000 00011001 (原码)

正数：25 00000000 00000000 00000000 00011001 (反码)

正数：25 00000000 00000000 00000000 00011001 (补码)

负数：-25 10000000 00000000 00000000 00011001 (原码)

负数：-25 11111111 11111111 11111111 11100110 (反码)

负数：-25 11111111 11111111 11111111 11100111 (补码)

一个字节可以存储的整数范围是多少？

//1 个字节：8 位

0000 0001 ~ 0111 1111 ==> 1~127

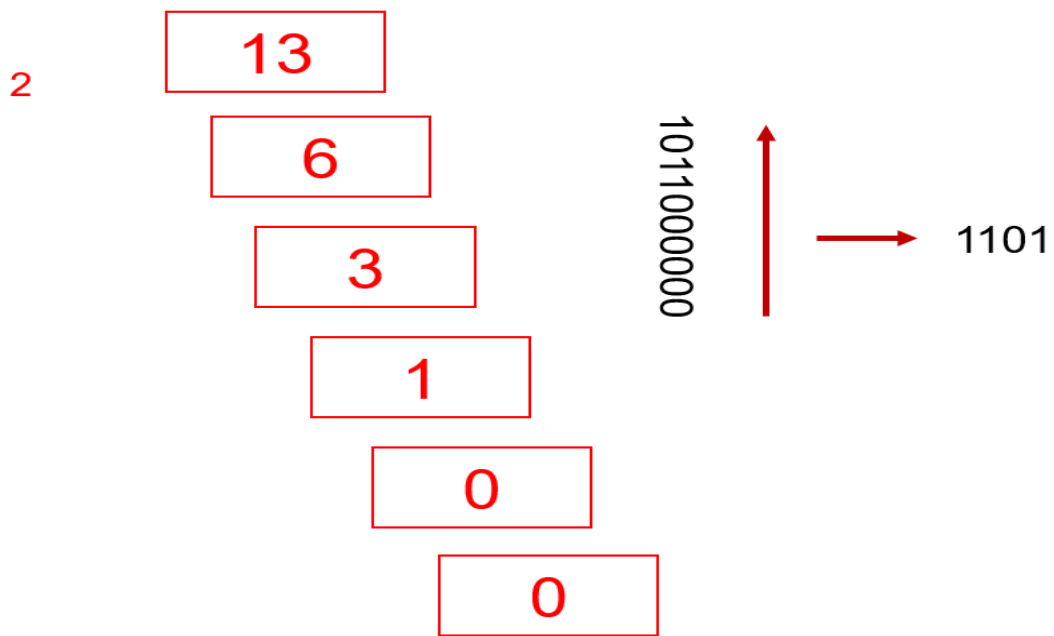
1000 0001 ~ 1111 1111 ==> -127 ~ -1

0000 0000 ==> 0

1000 0000 ==> -128 (特殊规定) = -127-1

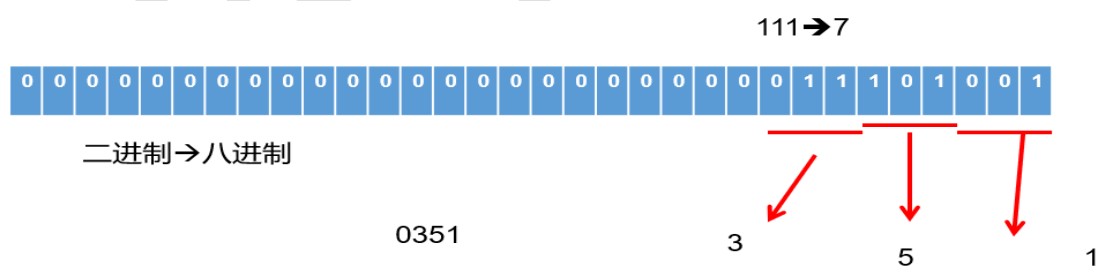
6.5 十进制转二进制

十进制转二进制：除 2 取余的逆

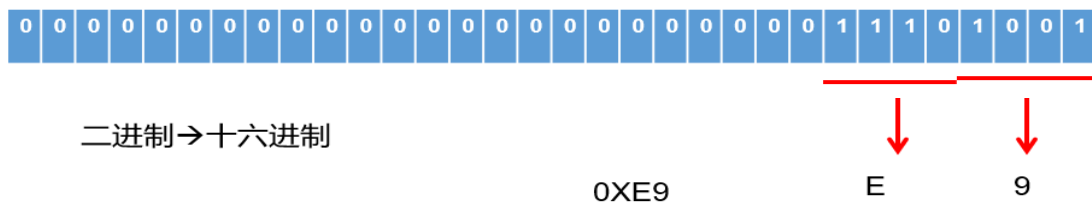


6.6 二进制与八进制、十六进制间的转换

二进制转八进制

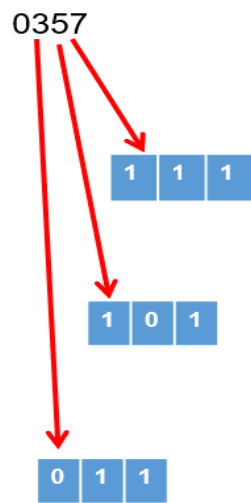


二进制转十六进制



八进制、十六进制转二进制

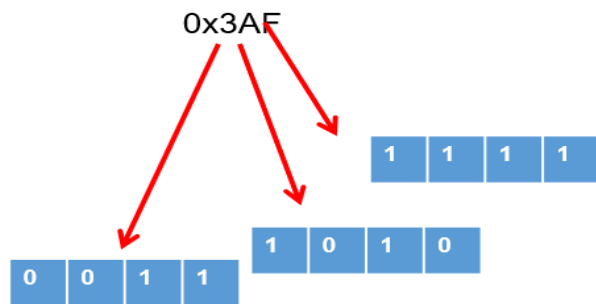
八进制:



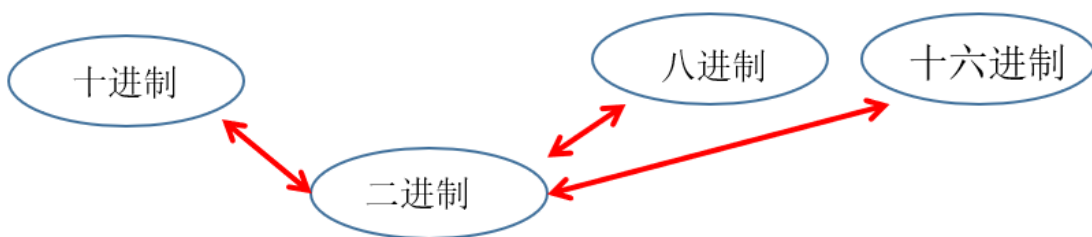
二进制:



十六进制



6.7 各进制间的转换



练习:

1.将以下十进制数转换为十六进制和二进制
123 256 87 62

2.将以下十六进制数转换为十进制和二进制
0x123 0x25F 0x38 0x62

7. 运算符（Operator）（掌握）

运算符是一种特殊的符号，用以表示数据的运算、赋值和比较等。

运算符的分类：

按照功能分为：算术运算符、赋值运算符、比较(或关系)运算符、逻辑运算符、位运算符、条件运算符、Lambda 运算符

分类	运算符
算术运算符（7 个）	+、-、*、/、%、++、--
赋值运算符（12 个）	=、+=、-=、*=、/=、%=、>>=、<<=、>>>=、&=、 =、^=等
比较(或关系)运算符（6 个）	>、>=、<、<=、==、!=
逻辑运算符（6 个）	&、 、^、!、&&、
位运算符（7 个）	&、 、^、~、<<、>>、>>>
条件运算符（1 个）	(条件表达式)?结果 1:结果 2
Lambda 运算符（1 个）	->（第 18 章时讲解）

按照操作数个数分为：

一元运算符（单目运算符）、二元运算符（双目运算符）、三元运算符（三目运算符）

分类	运算符
一元运算符（单目运算符）	正号（+）、负号（-）、++、--、!、~
二元运算符（双目运算符）	除了一元和三元运算符剩下的都是二元运算符
三元运算符（三目运算符）	(条件表达式)?结果 1:结果 2

7.1 算术运算符

7.1.1 基本语法

运算符	运算	范例	结果
+	正号	+3	3
-	负号	b=4; -b	-4
+	加	5+5	10
-	减	6-4	2
*	乘	3*4	12
/	除	5/5	1
%	取模(取余)	7%5	2
++	自增（前）：先运算后取值	a=2;b=++a;	a=3;b=3
++	自增（后）：先取值后运算	a=2;b=a++;	a=3;b=2
--	自减（前）：先运算后取值	a=2;b=- -a	a=1;b=1
--	自减（后）：先取值后运算	a=2;b=a- -	a=1;b=2
+	字符串连接	"He"+"llo"	"Hello"

举例 1：加减乘除模

```

public class ArithmeticTest1 {
    public static void main(String[] args) {
        int a = 3;
        int b = 4;

        System.out.println(a + b); // 7
        System.out.println(a - b); // -1
        System.out.println(a * b); // 12
        System.out.println(a / b); // 计算机结果是 0，为什么不是 0.75 呢？
        System.out.println(a % b); // 3

        // 结果与被模数符号相同
        System.out.println(5%2); // 1
    }
}

```

```

        System.out.println(5%-2); //1
        System.out.println(-5%2); // -1
        System.out.println(-5%-2); // -1
        // 商*除数 + 余数 = 被除数
        // 5%-2 ==> 商是-2, 余数是1    (-2)*(-2)+1 = 5
        // -5%2 ==> 商是-2, 余数是-1   (-2)*2+(-1) = -4-1=-5
    }
}

```

举例 2: “+”号的两种用法

第一种: 对于+两边都是数值的话, +就是加法的意思

第二种: 对于+两边至少有一边是字符串的话, +就是拼接的意思

```

public class ArithmeticTest2 {
    public static void main(String[] args) {
        // 字符串类型的变量基本使用
        // 数据类型 变量名称 = 数据值;
        String str1 = "Hello";
        System.out.println(str1); // Hello

        System.out.println("Hello" + "World"); // HelloWorld

        String str2 = "Java";
        // String + int --> String
        System.out.println(str2 + 520); // Java520
        // String + int + int
        // String      + int
        // String
        System.out.println(str2 + 5 + 20); // Java520
    }
}

```

举例 3: 自加自减运算

理解: ++ 运算, 表示自增1。同理, -- 运算, 表示自减1, 用法与++ 一致。

1、单独使用

- 变量在单独运算的时候, 变量前++和变量后++, 是没有区别的。
- 变量前++ : 例如 ++a 。

- 变量后++：例如 `a++`。

```
public class ArithmeticTest3 {
    public static void main(String[] args) {
        // 定义一个int 类型的变量a
        int a = 3;
        //++a;
        a++;
        // 无论是变量前++还是变量后++，结果都是4
        System.out.println(a);
    }
}
```

2、复合使用

- 和其他变量放在一起使用或者和输出语句放在一起使用，前++和后++就产生了不同。
- 变量前++：变量先自增1，然后再运算。
- 变量后++：变量先运算，然后再自增1。

```
public class ArithmeticTest4 {
    public static void main(String[] args) {
        // 其他变量放在一起使用
        int x = 3;
        //int y = ++x; // y 的值是4, x 的值是4,
        int y = x++; // y 的值是3, x 的值是4

        System.out.println(x);
        System.out.println(y);
        System.out.println("=====");

        // 和输出语句一起
        int z = 5;
        //System.out.println(++z);// 输出结果是6, z 的值也是6
        System.out.println(z++);// 输出结果是5, z 的值是6
        System.out.println(z);
    }
}
```


7.1.2 案例与练习

案例 1:

随意给出一个整数，打印显示它的个位数，十位数，百位数的值。

格式如下：

数字 xxx 的情况如下：

个位数：

十位数：

百位数：

例如：

数字 153 的情况如下：

个位数：3

十位数：5

百位数：1

```
/**
 * @author 尚硅谷-宋红康
 * @create 12:20
 */
class ArithmeticExer1 {
    public static void main(String[] args) {

        int num = 187;

        int bai = num / 100;
        int shi = num % 100 / 10; //int shi = num / 10 % 10;
        int ge = num % 10;

        System.out.println("百位为: " + bai);
        System.out.println("十位为: " + shi);
        System.out.println("个位为: " + ge);

    }
}
```

拓展：获取一个四位数的个位，十位，百位，千位

```
public class ArithmeticExer01 {
    public static void main (String [] args) {
        //1. 定义一个变量，赋值为一个四位数整数，例如 1234
```

```

int num = 1234;

//2. 通过运算操作求出个位, 十位, 百位, 千位
int ge = num % 10;
int shi = num / 10 % 10;
int bai = num / 100 % 10;
int qian = num / 1000 % 10;

System.out.println("个位上的数字是: " + ge);
System.out.println("十位上的数字是: " + shi);
System.out.println("百位上的数字是: " + bai);
System.out.println("千位上的数字是: " + qian);
}
}

```

案例 2: 为抵抗洪水, 战士连续作战 89 小时, 编程计算共多少天零多少小时?

```

public class ArithmeticExer2 {
    public static void main(String[] args){
        int hours = 89;
        int day = hours / 24;
        int hour = hours % 24;
        System.out.println("为抵抗洪水, 战士连续作战 89 小时: ");
        System.out.println(hours + "是" + day + "天" + hour + "小时");
    }
}

```

练习 1: 算术运算符: 自加、自减

```

public class ArithmeticExer3{
    public static void main(String[] args){
        int i1 = 10;
        int i2 = 20;
        int i = i1++;
        System.out.print("i="+i); //
        System.out.println("i1="+i1);//
        i = ++i1;
        System.out.print("i="+i);//
        System.out.println("i1="+i1);//
        i = i2--;
        System.out.print("i="+i);//
        System.out.println("i2="+i2);//
        i = --i2;
        System.out.print("i="+i);//
        System.out.println("i2="+i2);//
    }
}

```

```
}  
}
```

练习 2:

```
System.out.println("5+5=" + 5 + 5); //打印结果是? 5+5=55 ?
```

练习 3:

```
byte bb1 = 127;  
bb1++;  
System.out.println("bb1 = " + bb1); //-128
```

练习 4:

```
int i = 1;  
int j = i++ + ++i * i++;  
System.out.println("j = " + j);
```

练习 5: (企业真题) 写出下列程序的输出结果

```
int i = 2;  
int j = i++;  
System.out.println(j);  
  
int m = 2;  
m = m++; //(1)先取b的值“2”放操作数栈 (2)m再自增,m=3 (3)再把操作数栈中的  
“2”赋值给m,m=2  
System.out.println(m);
```

7.2 赋值运算符

7.2.1 基本语法

符号: =

- 当“=”两侧数据类型不一致时, 可以使用自动类型转换或使用强制类型转换原则进行处理。
- 支持连续赋值。

扩展赋值运算符: +=、 -=、 *=、 /=、 %=

赋值运

算符

符号解释

+=

将符号左边的值和右边的值进行相加操作，最后将结果赋值给左边的变量

-=

将符号左边的值和右边的值进行相减操作，最后将结果赋值给左边的变量

*=

将符号左边的值和右边的值进行相乘操作，最后将结果赋值给左边的变量

/=

将符号左边的值和右边的值进行相除操作，最后将结果赋值给左边的变量

%=

将符号左边的值和右边的值进行取余操作，最后将结果赋值给左边的变量

```
public class SetValueTest1 {  
    public static void main(String[] args) {  
        int i1 = 10;  
        long l1 = i1; //自动类型转换  
  
        byte bb1 = (byte)i1; //强制类型转换  
  
        int i2 = i1;  
  
        //连续赋值的测试  
        //以前的写法  
        int a1 = 10;  
        int b1 = 10;  
  
        //连续赋值的写法  
        int a2, b2;  
        a2 = b2 = 10;  
  
        int a3 = 10, b3 = 20;
```

```

//举例说明+= -= *= /= %=
int m1 = 10;
m1 += 5; //类似于 m1 = m1 + 5 的操作, 但不等同于。
System.out.println(m1);//15

//练习1: 开发中, 如何实现一个变量+2 的操作呢?
// += 的操作不会改变变量本身的数据类型。其他拓展的运算符也如此。
//写法1: 推荐
short s1 = 10;
s1 += 2; //编译通过, 因为在得到int 类型的结果后, JVM 自动完成一步
强制类型转换, 将int 类型强转成 short
System.out.println(s1);//12
//写法2:
short s2 = 10;
//s2 = s2 + 2; //编译报错, 因为将int 类型的结果赋值给 short 类型的
变量s 时, 可能损失精度
s2 = (short)(s2 + 2);
System.out.println(s2);

//练习2: 开发中, 如何实现一个变量+1 的操作呢?
//写法1: 推荐
int num1 = 10;
num1++;
System.out.println(num1);

//写法2:
int num2 = 10;
num2 += 1;
System.out.println(num2);

//写法3:
int num3 = 10;
num3 = num3 + 1;
System.out.println(num3);

}
}

```

7.2.2 练习

练习 1:

```
short s = 3;
s = s+2; //① 编译报错
s += 2; //② 正常执行
```

//①和②有什么区别?

练习 2:

```
int i = 1;
i *= 0.1;
System.out.println(i); //0
i++;
System.out.println(i); //1
```

练习 3:

```
int m = 2;
int n = 3;
n *= m++; //n = n * m++;
System.out.println("m=" + m); //3
System.out.println("n=" + n); //6
```

练习 4:

```
int n = 10;
n += (n++) + (++n); //n = n + (n++) + (++n)
System.out.println(n); //32
```

练习 5: 你有几种办法实现变量值减 1? 变量值减 2 呢?

```
/**
 * @author 尚硅谷-宋红康
 * @create 16:55
 */
public class MinusTest {
    public static void main(String[] args) {
        //练习①: 变量值减 1
        short s = 10;
        //方式 1:
        //s = (short)(s - 1);
        //方式 2: 推荐
        s--; //或者 --s
        //方式 3:
```

```

        s -= 1;

        //练习②: 变量值减 2
        short s1 = 10;
        //方式 1:
        //s1 = (short)(s1 - 2);
        //方式 2: 推荐
        s1 -= 2;
    }
}

```

7.3 比较(关系)运算符

运算符	运算	范例	结果
==	相等于	4==3	false
!=	不等于	4!=3	true
<	小于	4<3	false
>	大于	4>3	true
<=	小于等于	4<=3	false
>=	大于等于	4>=3	true
instanceof	检查是否是类的对象	"Hello" instanceof String	true

- 比较运算符的结果都是 boolean 型，也就是要么是 true，要么是 false。
- > < >= <= : 只适用于基本数据类型（除 boolean 类型之外）
 == != : 适用于基本数据类型和引用数据类型
- 比较运算符“==”不能误写成“=”

举例：

```

class CompareTest {
    public static void main(String[] args) {
        int i1 = 10;
        int i2 = 20;

        System.out.println(i1 == i2); //false
        System.out.println(i1 != i2); //true
    }
}

```

```

        System.out.println(i1 >= i2);//false

        int m = 10;
        int n = 20;
        System.out.println(m == n);//false
        System.out.println(m = n);//20

        boolean b1 = false;
        boolean b2 = true;
        System.out.println(b1 == b2);//false
        System.out.println(b1 = b2);//true
    }
}

```

思考：

```

boolean b1 = false;
//区分好==和=的区别。
if(b1 == true) //if(b1 = true)
    System.out.println("结果为真");
else
    System.out.println("结果为假");

```

7.4 逻辑运算符

7.4.1 基本语法

a	b	a&b	a&&b	a b	a b	!a	a^b
true	true	true	true	true	true	false	false
true	false	false	false	true	true	false	true
false	true	false	false	true	true	true	true
false	false	false	false	false	false	true	false

逻辑运算符，操作的都是 boolean 类型的变量或常量，而且运算得结果也是 boolean 类型的值。

运算符说明：

- & 和 &&：表示"且"关系，当符号左右两边布尔值都是 true 时，结果才能为 true。否则，为 false。
- | 和 ||：表示"或"关系，当符号两边布尔值有一边为 true 时，结果为 true。当两边都为 false 时，结果为 false
- !：表示"非"关系，当变量布尔值为 true 时，结果为 false。当变量布尔值为 false 时，结果为 true。
- ^：当符号左右两边布尔值不同时，结果为 true。当两边布尔值相同时，结果为 false。
 - 理解：异或，追求的是“异”！

逻辑运算符用于连接布尔型表达式，在 Java 中不可以写成 $3 < x < 6$ ，应该写成 $x > 3 \& x < 6$ 。

区分"&"和"&&"：

- 相同点：如果符号左边是 true，则二者都执行符号右边的操作
- 不同点：&：如果符号左边是 false,则继续执行符号右边的操作
&&：如果符号左边是 false,则不再继续执行符号右边的操作
- 建议：开发中，推荐使用 &&

区分"|"和"||"：

- 相同点：如果符号左边是 false，则二者都执行符号右边的操作
- 不同点：|：如果符号左边是 true，则继续执行符号右边的操作
||：如果符号左边是 true，则不再继续执行符号右边的操作
- 建议：开发中，推荐使用 ||

代码举例：

```
public class LoginTest {
    public static void main(String[] args) {
        int a = 3;
        int b = 4;
        int c = 5;

        // & 与, 且; 有false 则false
        System.out.println((a > b) & (a > c));
        System.out.println((a > b) & (a < c));
        System.out.println((a < b) & (a > c));
    }
}
```

```

System.out.println((a < b) & (a < c));
System.out.println("=====");
// | 或; 有true 则true
System.out.println((a > b) | (a > c));
System.out.println((a > b) | (a < c));
System.out.println((a < b) | (a > c));
System.out.println((a < b) | (a < c));
System.out.println("=====");
// ^ 异或; 相同为false, 不同为true
System.out.println((a > b) ^ (a > c));
System.out.println((a > b) ^ (a < c));
System.out.println((a < b) ^ (a > c));
System.out.println((a < b) ^ (a < c));
System.out.println("=====");
// ! 非; 非false 则true, 非true 则false
System.out.println(!false);
System.out.println(!true);

//&和&&的区别
System.out.println((a > b) & (a++ > c));
System.out.println("a = " + a);
System.out.println((a > b) && (a++ > c));
System.out.println("a = " + a);
System.out.println((a == b) && (a++ > c));
System.out.println("a = " + a);

//|和||的区别
System.out.println((a > b) | (a++ > c));
System.out.println("a = " + a);
System.out.println((a > b) || (a++ > c));
System.out.println("a = " + a);
System.out.println((a == b) || (a++ > c));
System.out.println("a = " + a);
}
}

```

7.4.2 案例与练习

案例：

1. 定义类 CompareLogicExer
2. 定义 main 方法
3. 定义一个 int 类型变量 a, 变量 b, 都赋值为 20

4. 定义 boolean 类型变量 bo1 , 判断++a 是否被 3 整除,并且 a++ 是否被 7 整除,将结果赋值给 bo1
5. 输出 a 的值,bo1 的值
6. 定义 boolean 类型变量 bo2 , 判断 b++ 是否被 3 整除,并且 ++b 是否被 7 整除,将结果赋值给 bo2
7. 输出 b 的值,bo2 的值

```
public class CompareLogicExer {  
    public static void main(String[] args){  
        int a = 20;  
        int b = 20;  
        boolean bo1 = ((++a % 3) == 0) && ((a++ % 7) == 0);  
        System.out.println("bo1 的值: " + bo1);  
        System.out.println("a 的值: " + a);  
        System.out.println("-----");  
  
        boolean bo2 = ((b++ % 3) == 0) && ((++b % 7) == 0);  
        System.out.println("bo2 的值: " + bo2);  
        System.out.println("b 的值: " + b);  
    }  
}
```

练习 1: 区分 & 和 &&

```
int x = 1;  
int y = 1;  
  
if(x++ == 2 & ++y == 2){  
    x = 7;  
}  
System.out.println("x=" + x + ",y=" + y);  
  
int x = 1,y = 1;  
  
if(x++ == 2 && ++y == 2){  
    x = 7;  
}  
System.out.println("x="+x+",y="+y);
```

练习 2: 区分 | 和 ||

```
int x = 1,y = 1;  
if(x++==1 | ++y==1){  
    x = 7;  
}
```

```
}  
System.out.println("x="+x+",y="+y);  
  
int x = 1,y = 1;  
if(x++==1 || ++y==1){  
    x =7;  
}  
System.out.println("x="+x+",y="+y);
```

练习 3：程序输出

```
class Test {  
    public static void main (String [] args) {  
        boolean x = true;  
        boolean y = false;  
        short z = 42;  
  
        if ((z++ == 42) && (y = true)) {  
            z++;  
        }  
        if ((x = false) || (++z == 45)) {  
            z++;  
        }  
        System.out.println("z=" + z);  
    }  
}
```

//结果为: //z= 46

7.5 位运算符（难点、非重点）

7.5.1 基本语法

位运算符			注意：无<<<
运算符	运算	范例	
<<	左移	3 << 2 = 12 --> 3*2*2=12	
>>	右移	3 >> 1 = 1 --> 3/2=1	
>>>	无符号右移	3 >>> 1 = 1 --> 3/2=1	
&	与运算	6 & 3 = 2	
	或运算	6 3 = 7	
^	异或运算	6 ^ 3 = 5	
~	取反运算	~6 = -7	

位运算符的细节	
<<	空位补0，被移除的高位丢弃，空缺位补0。
>>	被移位的二进制最高位是0，右移后，空缺位补0；最高位是1，空缺位补1。
>>>	被移位二进制最高位无论是0或者是1，空缺位都用0补。
&	二进制位进行&运算，只有1&1时结果是1，否则是0；
	二进制位进行 运算，只有0 0时结果是0，否则是1；
^	相同二进制位进行^运算，结果是0；1^1=0，0^0=0 不相同二进制位^运算结果是1。1^0=1，0^1=1
~	正数取反，各二进制码按补码各位取反 负数取反，各二进制码按补码各位取反

位运算符的运算过程都是基于二进制的补码运算

(1) 左移：<<

运算规则：在一定范围内，数据每向左移动一位，相当于原数据*2。（正数、负数都适用）

【注意】当左移的位数 n 超过该数据类型的总位数时，相当于左移 $(n - \text{总位数})$ 位

$3 \ll 4$ 类似于 $3 * 2$ 的 4 次幂 $\Rightarrow 3 * 16 \Rightarrow 48$

```
/*
3的二进制: 0000 0000 0000 0000 0000 0000 0000 0011
3<<4: 0000 0000 0000 0000 0000 0000 0000 0011 0000
*/
```

左边移出去4位, 右边补4个0

$-3 \ll 4$ 类似于 $-3 * 2$ 的 4 次幂 $\Rightarrow -3 * 16 \Rightarrow -48$

```
/*
-3的二进制:
原码: 1000 0000 0000 0000 0000 0000 0000 0011
反码: 1111 1111 1111 1111 1111 1111 1111 1100
补码: 1111 1111 1111 1111 1111 1111 1111 1101
-3<<4: 1111 1111 1111 1111 1111 1111 1101 0000
补码: 1111 1111 1111 1111 1111 1111 1101 0000
反码: 1111 1111 1111 1111 1111 1111 1100 1111
原码: 1000 0000 0000 0000 0000 0000 0011 0000
*/
```

左边移出去4位, 右边补4个0

运算用补码, 看结果用原码

(2) 右移: \gg

运算规则: 在一定范围内, 数据每向右移动一位, 相当于原数据/2。(正数、负数都适用)

【注意】如果不能整除, 向下取整。

$69 \gg 4$ 类似于 $69 / 2$ 的 4 次 $= 69 / 16 = 4$

```

/*
69的二进制: 0000 0000 0000 0000 0000 0000 0100 0101
69>>4:       0000 0000 0000 0000 0000 0000 0000 0100 0101
*/           正数左边补4个0                      右边移出去4位

```

-69>>4 类似于 $-69/2$ 的 4 次 = $-69/16 = -5$

```

/*
-69的二进制:
补码: 1000 0000 0000 0000 0000 0000 0100 0101
反码: 1111 1111 1111 1111 1111 1111 1011 1010
补码: 1111 1111 1111 1111 1111 1111 1011 1011
-69>>4:       1111 1111 1111 1111 1111 1111 1111 1011 1011
负数左边      补码: 1111 1111 1111 1111 1111 1111 1111 1011 右边移出去
补4个1        反码: 1111 1111 1111 1111 1111 1111 1111 1010 4位
              原码: 1000 0000 0000 0000 0000 0000 0000 0101
*/

```

(3) 无符号右移: >>>

运算规则: 往右移动后, 左边空出来的位直接补 0。(正数、负数都适用)

69>>>4 类似于 $69/2$ 的 4 次 = $69/16 = 4$

```

/*
69的二进制: 0000 0000 0000 0000 0000 0000 0100 0101
69>>>4:       0000 0000 0000 0000 0000 0000 0000 0100 0101
*/           左边补4个0                      右边移出去4位

```

-69>>>4 结果: 268435451

/*

-69的二进制:

补码: 1000 0000 0000 0000 0000 0000 0100 0101

反码: 1111 1111 1111 1111 1111 1111 1011 1010

补码: 1111 1111 1111 1111 1111 1111 1011 1011

-69>>>4:

0000 1111 1111 1111 1111 1111 1111 1011 1011

无符号右移
左边补4个0

补码: 0000 1111 1111 1111 1111 1111 1111 1011

右边移出
去4位

反码: 0000 1111 1111 1111 1111 1111 1111 1011

原码: 0000 1111 1111 1111 1111 1111 1111 1011

最高位是0, 是正数, 那么原码, 反码, 补码一样

*/

计算用补码, 看结果用原码

(4) 按位与: &

运算规则: 对应位都是 1 才为 1, 否则为 0。

- 1 & 1 结果为 1
- 1 & 0 结果为 0
- 0 & 1 结果为 0
- 0 & 0 结果为 0

9 & 7 = 1

/*

9的二进制: 0000 0000 0000 0000 0000 0000 0000 1001

7的二进制: 0000 0000 0000 0000 0000 0000 0000 0111

9&7: 0000 0000 0000 0000 0000 0000 0000 0001

*/

&

-9 & 7 = 7

/*

-9的二进制:

原码: 1000 0000 0000 0000 0000 0000 0000 1001

反码: 1111 1111 1111 1111 1111 1111 1111 0110

补码: 1111 1111 1111 1111 1111 1111 1111 0111

7的二进制: 0000 0000 0000 0000 0000 0000 0000 0111

&

-9&7: 0000 0000 0000 0000 0000 0000 0000 0111

补码: 0000 0000 0000 0000 0000 0000 0000 0111

反码: 0000 0000 0000 0000 0000 0000 0000 0111

原码: 0000 0000 0000 0000 0000 0000 0000 0111

*/

(5) 按位或: |

运算规则: 对应位只要有 1 即为 1, 否则为 0。

- 1 | 1 结果为 1
- 1 | 0 结果为 1
- 0 | 1 结果为 1
- 0 & 0 结果为 0

9 | 7 //结果: 15

/*

9的二进制: 0000 0000 0000 0000 0000 0000 0000 1001

7的二进制: 0000 0000 0000 0000 0000 0000 0000 0111

9 | 7: 0000 0000 0000 0000 0000 0000 0000 1111

*/

-9 | 7 //结果: -9

```

/*
-9的二进制:
    原码: 1000 0000 0000 0000 0000 0000 0000 1001
    反码: 1111 1111 1111 1111 1111 1111 1111 0110
    补码: 1111 1111 1111 1111 1111 1111 1111 0111
7的二进制: 0000 0000 0000 0000 0000 0000 0000 0111
-9|7:      1111 1111 1111 1111 1111 1111 1111 0111
    补码: 1111 1111 1111 1111 1111 1111 1111 0111
    反码: 1111 1111 1111 1111 1111 1111 1111 0110
    原码: 1000 0000 0000 0000 0000 0000 0000 1001
*/

```

(6) 按位异或: ^

运算规则: 对应位一个为 1 一个为 0, 才为 1, 否则为 0。

- $1 \wedge 1$ 结果为 0
- $1 \wedge 0$ 结果为 1
- $0 \wedge 1$ 结果为 1
- $0 \wedge 0$ 结果为 0

$9 \wedge 7$ // 结果为 14

```

/*
9的二进制: 0000 0000 0000 0000 0000 0000 0000 1001
7的二进制: 0000 0000 0000 0000 0000 0000 0000 0111
9 ^ 7:      0000 0000 0000 0000 0000 0000 0000 1110
*/

```

$-9 \wedge 7$ // 结果为 -16

```

/*
-9的二进制:
    原码: 1000 0000 0000 0000 0000 0000 0000 1001
    反码: 1111 1111 1111 1111 1111 1111 1111 0110
    补码: 1111 1111 1111 1111 1111 1111 1111 0111
7的二进制: 0000 0000 0000 0000 0000 0000 0000 0111 ^
-9^7:      1111 1111 1111 1111 1111 1111 1111 0000
    补码: 1111 1111 1111 1111 1111 1111 1111 0000
    反码: 1111 1111 1111 1111 1111 1111 1110 1111
    原码: 1000 0000 0000 0000 0000 0000 0001 0000
*/

```

(7) 按位取反: ~

运算规则: 对应位为 1, 则结果为 0; 对应位为 0, 则结果为 1。

- ~0 就是 1
- ~1 就是 0

~9 // 结果: -10

```

/*
9的二进制: 0000 0000 0000 0000 0000 0000 0000 1001 取反
~9:        1111 1111 1111 1111 1111 1111 1111 0110
    补码: 1111 1111 1111 1111 1111 1111 1111 0110
    反码: 1111 1111 1111 1111 1111 1111 1111 0101
    原码: 1000 0000 0000 0000 0000 0000 0000 1010
*/

```

~-9 // 结果: 8

/*

-9的二进制:

原码: 1000 0000 0000 0000 0000 0000 0000 1001

反码: 1111 1111 1111 1111 1111 1111 1111 0110

补码: 1111 1111 1111 1111 1111 1111 1111 0111 取反

~-9: 0000 0000 0000 0000 0000 0000 0000 1000

补码: 0000 0000 0000 0000 0000 0000 0000 1000

反码: 0000 0000 0000 0000 0000 0000 0000 1000

原码: 0000 0000 0000 0000 0000 0000 0000 1000

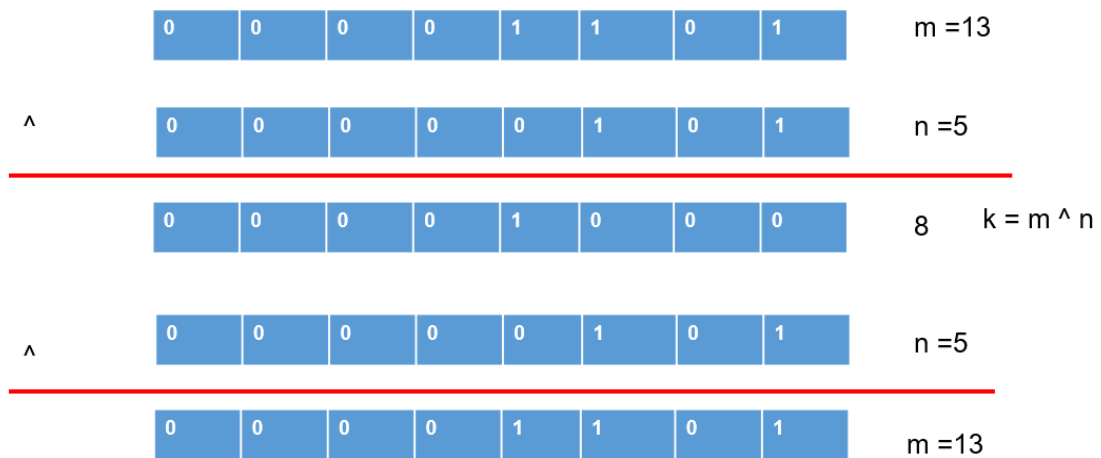
*/

7.5.2 举例

举例 1:

	0	0	0	0	1	1	0	0	12
&	0	0	0	0	0	1	0	1	5
	0	0	0	0	0	1	0	0	4
	0	0	0	0	1	1	0	0	12
	0	0	0	0	0	1	0	1	5
	0	0	0	0	1	1	0	1	13
	0	0	0	0	1	1	0	0	12
^	0	0	0	0	0	1	0	1	5
	0	0	0	0	1	0	0	1	9

举例 2：体会 $m = k \wedge n = (m \wedge n) \wedge n$



7.5.3 案例

案例 1：高效的方式计算 $2 * 8$ 的值（经典面试题）

答案： $2 \ll 3$ 、 $8 \ll 1$

案例 2：如何交换两个 int 型变量的值？String 呢？

```
/**
 * @author 尚硅谷-宋红康
 * @create 16:58
 */
public class BitExer {
    public static void main(String[] args) {
        int m = 10;
        int n = 5;

        System.out.println("m = " + m + ", n = " + n);
```

//（推荐）实现方式 1：优点：容易理解，适用于不同数据类型 缺点：需要额外定义变量

```
//int temp = m;
//m = n;
//n = temp;
```

//实现方式 2：优点：没有额外定义变量 缺点：可能超出 int 的范围；只能适用于数值类型

```
//m = m + n; //15 = 10 + 5
//n = m - n; //10 = 15 - 5
//m = m - n; //5 = 15 - 10
```

//实现方式3: 优点: 没有额外定义变量 缺点: 不易理解; 只能适用于数值类型

```
m = m ^ n;
n = m ^ n; //(m ^ n) ^ n
m = m ^ n;

System.out.println("m = " + m + ", n = " + n);
}
}
```

7.6 条件运算符

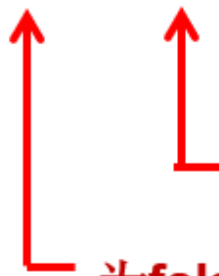
7.6.1 基本语法

条件运算符格式:

(条件表达式)? 表达式 1: 表达式 2

说明: 条件表达式是 boolean 类型的结果, 根据 boolean 的值选择表达式 1 或表达式 2

(条件表达式)?表达式1: 表达式2;

 **为true**, 运算后的结果是表达式1;
为false, 运算后的结果是表达式2;

如果运算后的结果赋给新的变量, 要求表达式 1 和表达式 2 为同种或兼容的类型

```
public static void main(String[] args) {
    int i = (1==2 ? 100 : 200);
    System.out.println(i); //200

    boolean marry = false;
    System.out.println(marry ? "已婚" : "未婚" );
}
```

```

double d1 = (m1 > m2)? 1 : 2.0;
System.out.println(d1);

int num = 12;
System.out.println(num > 0? true : "num 非正数");
}

```

7.6.2 案例

案例 1： 获取两个数中的较大值

```

public class ConditionExer1 {
    public static void main(String[] args) {
        // 获取两个数的较大值
        int m1 = 10;
        int m2 = 20;

        int max1 = (m1 > m2)? m1 : m2;
        System.out.println("m1 和 m2 中的较大值为" + max1);
    }
}

```

案例 2： 获取三个数中的最大值

```

public class ConditionExer2 {
    public static void main(String[] args) {
        int n1 = 23;
        int n2 = 13;
        int n3 = 33;
        // 写法1:
        int tempMax = (n1 > n2)? n1:n2;
        int finalMax = (tempMax > n3)? tempMax : n3;
        System.out.println("三个数中最大值为: " + finalMax);

        // 写法2: 不推荐, 可读性差
        int finalMax1 = (((n1 > n2)? n1:n2) > n3)? ((n1 > n2)? n1:n2)
: n3;
        System.out.println("三个数中最大值为: " + finalMax1);
    }
}

```

案例 3：今天是周 2，10 天以后是周几？

要求：控制台输出"今天是周 2，10 天以后是周 x"。

```
public class ConditionExer3 {  
  
    public static void main(String[] args) {  
        int week = 2;  
        week += 10;  
        week %= 7;  
        System.out.println("今天是周 2,10 天以后是周" + (week == 0 ? "日"  
" : week));  
    }  
}
```

7.6.3 与 if-else 的转换关系

凡是可以使用条件运算符的地方，都可以改写为 if-else 结构。反之，不成立。

开发中，如果既可以使用条件运算符，又可以使用 if-else，推荐使用条件运算符。因为执行效率稍高。

//if-else 实现获取两个数的较大值

```
int i1 = 10;  
int i2 = 20;  
  
int max;//声明变量max, 用于记录i1和i2的较大值  
  
if(i1 > i2){  
    max = i1;  
}  
else{  
    max = i2;  
}  
  
System.out.println(max);
```

7.7 运算符优先级

运算符有不同的优先级，所谓优先级就是在表达式运算中的运算符顺序。

上一行中的运算符总是优先于下一行的。

优先级	运算符说明	Java 运算符
1	括号	<code>()</code> 、 <code>[]</code> 、 <code>{}</code>
2	正负号	<code>+</code> 、 <code>-</code>
3	单元运算符	<code>++</code> 、 <code>--</code> 、 <code>~</code> 、 <code>!</code>
4	乘法、除法、求余	<code>*</code> 、 <code>/</code> 、 <code>%</code>
5	加法、减法	<code>+</code> 、 <code>-</code>
6	移位运算符	<code><<</code> 、 <code>>></code> 、 <code>>>></code>
7	关系运算符	<code><</code> 、 <code><=</code> 、 <code>>=</code> 、 <code>></code> 、 <code>instanceof</code>
8	等价运算符	<code>==</code> 、 <code>!=</code>
9	按位与	<code>&</code>
10	按位异或	<code>^</code>
11	按位或	<code> </code>
12	条件与	<code>&&</code>
13	条件或	<code>//</code>
14	三元运算符	<code>?</code> <code>:</code>
15	赋值运算符	<code>=</code> 、 <code>+=</code> 、 <code>-=</code> 、 <code>*=</code> 、 <code>/=</code> 、 <code>%=</code>
16	位赋值运算符	<code>&=</code> 、 <code>/=</code> 、 <code><<=</code> 、 <code>>>=</code> 、 <code>>>>=</code>

开发建议：

1. 不要过多的依赖运算的优先级来控制表达式的执行顺序，这样可读性太差，尽量使用`()`来控制表达式的执行顺序。
2. 不要把一个表达式写得过于复杂，如果一个表达式过于复杂，则把它分成几步来完成。例如：
`(num1 + num2) * 2 > num3 &&`
`num2 > num3 ? num3 : num1 + num2;`

8. 【拓展】关于字符集

8.1 字符集

编码与解码：

计算机中储存的信息都是用二进制数表示的，而我们在屏幕上看到的数字、英文、标点符号、汉字等字符是二进制数转换之后的结果。按照某种规则，将字符存储到计算机中，称为**编码**。反之，将存储在计算机中的二进制数按照某种规则解析显示出来，称为**解码**。

字符编码 (Character Encoding)：就是一套自然语言的字符与二进制数之间的对应规则。

字符集：也叫编码表。是一个系统支持的所有字符的集合，包括各国家文字、标点符号、图形符号、数字等。

8.2 ASCII 码

ASCII 码 (American Standard Code for Information Interchange, 美国信息交换标准代码)：上个世纪 60 年代，美国制定了一套字符编码，对英语字符与二进制位之间的关系，做了统一规定。这被称为 ASCII 码。

ASCII 码用于显示现代英语，主要包括控制字符（回车键、退格、换行键等）和可显示字符（英文大小写字符、阿拉伯数字和西文符号）。

基本的 ASCII 字符集，使用 7 位 (bits) 表示一个字符（最前面的 1 位统一规定为 0），共 128 个字符。比如：空格“SPACE”是 32（二进制 00100000），大写的字母 A 是 65（二进制 01000001）。

缺点：不能表示所有字符。

ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符
0	NUL	32	(space)	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(72	H	104	h
9	HT	41)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

8.3 ISO-8859-1 字符集

拉丁码表，别名 Latin-1，用于显示欧洲使用的语言，包括荷兰语、德语、意大利语、葡萄牙语等

ISO-8859-1 使用单字节编码，兼容 ASCII 编码。

8.4 GBxxx 字符集

GB 就是国标的意，是为了显示中文而设计的一套字符集。

GB2312：简体中文码表。一个小于 127 的字符的意义与原来相同，即向下兼容 ASCII 码。但两个大于 127 的字符连在一起时，就表示一个汉字，这样大约可以组合了包含 7000 多个简体汉字，此外数学符号、罗马希腊的字母、日文的假名们都编进去了，这就是常说的"全角"字符，而原来在 127 号以下的那些符号就叫"半角"字符了。

GBK：最常用的中文码表。是在 GB2312 标准基础上的扩展规范，使用了双字节编码方案，共收录了 21003 个汉字，完全兼容 GB2312 标准，同时支持繁体汉字以及日韩汉字等。

GB18030：最新的中文码表。收录汉字 70244 个，采用多字节编码，每个字可以由 1 个、2 个或 4 个字节组成。支持中国国内少数民族的文字，同时支持繁体汉字以及日韩汉字等。

8.5 Unicode 码

Unicode 编码为表达*任意语言的任意字符*而设计，也称为统一码、标准万国码。

Unicode 将世界上所有的文字用 **2 个字节**统一进行编码，为每个字符设定唯一的二进制编码，以满足跨语言、跨平台进行文本处理的要求。

Unicode 的缺点：这里有三个问题：

- 第一，英文字母只用一个字节表示就够了，如果用更多的字节存储是*极大的浪费*。
- 第二，如何才能*区别 Unicode 和 ASCII*？计算机怎么知道两个字节表示一个符号，而不是分别表示两个符号呢？
- 第三，如果和 GBK 等双字节编码方式一样，用最高位是 1 或 0 表示两个字节和一个字节，就少了很多值无法用于表示字符，*不够表示所有字符*。

Unicode 在很长一段时间内无法推广，直到互联网的出现，为解决 Unicode 如何在网络上传输的问题，于是面向传输的众多 UTF (UCS Transfer Format) 标准出现。具体来说，有三种编码方案，UTF-8、UTF-16 和 UTF-32。

8.6 UTF-8

Unicode 是字符集，UTF-8、UTF-16、UTF-32 是三种*将数字转换到程序数据*的编码方案。顾名思义，UTF-8 就是每次 8 个位传输数据，而 UTF-16 就是每次 16 个位。其中，UTF-8 是在互联网上*使用最广*的一种 Unicode 的实现方式。

互联网工程工作小组 (IETF) 要求所有互联网协议都必须支持 UTF-8 编码。所以，我们开发 Web 应用，也要使用 UTF-8 编码。UTF-8 是一种*变长的编码方式*。它可以使用 1-4 个字节表示一个符号它使用一至四个字节为每个字符编码，编码规则：

1. 128 个 US-ASCII 字符，只需一个字节编码。
2. 拉丁文等字符，需要二个字节编码。
3. 大部分常用字（含中文），使用三个字节编码。
4. 其他极少使用的 Unicode 辅助字符，使用四字节编码。

举例：

Unicode 符号范围 | UTF-8 编码方式

(十六进制) | (二进制)

_____ | _____

0000 0000-0000 007F | 0xxxxxxx (兼容原来的 ASCII)

0000 0080-0000 07FF | 110xxxxx 10xxxxxx

0000 0800-0000 FFFF | 1110xxxx 10xxxxxx 10xxxxxx

0001 0000-0010 FFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

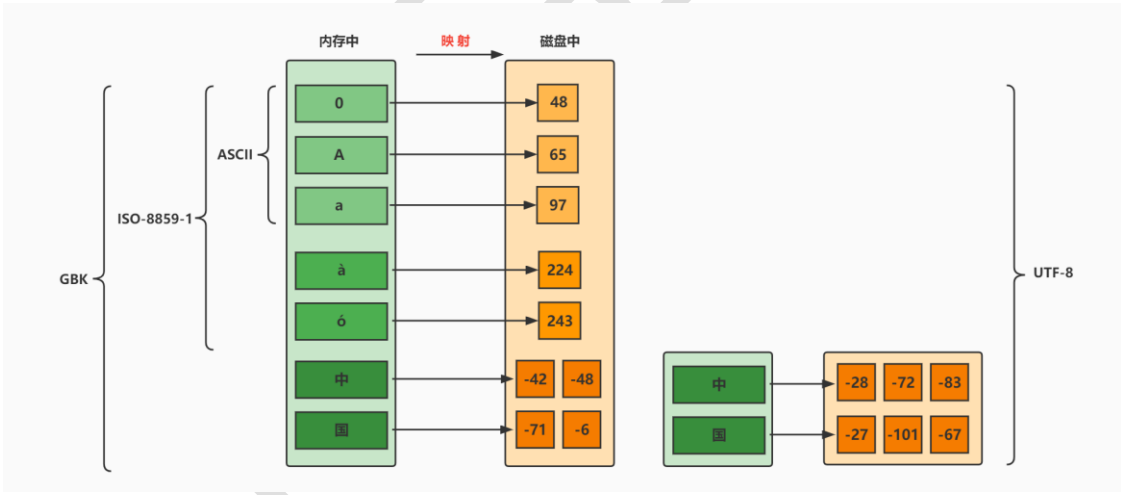
尚

Unicode编码值：23578 十六进制 5C1A 二进制 0101 1100 0001 1010

1110xxxx 10xx xxxx 10xx xxxx
1110 0101 1011 0000 1001 1010

UTF-8编码： 1110 0101 1011 0000 1001 1010 e5 b0 9a [-27, -80, -102]

8.7 小结



注意：在中文操作系统上，ANSI（美国国家标准学会、AMERICAN NATIONAL STANDARDS INSTITUTE: ANSI）编码即为 GBK；在英文操作系统上，ANSI 编码即为 ISO-8859-1。

本章专题与脉络



第 1 阶段: Java 基本语法-第 03 章

流程控制语句是用来控制程序中各语句执行顺序的语句，可以把语句组合成能完成一定功能的小逻辑模块。

程序设计中规定的三种流程结构，即：

顺序结构

- 程序从上到下逐行地执行，中间没有任何判断和跳转。

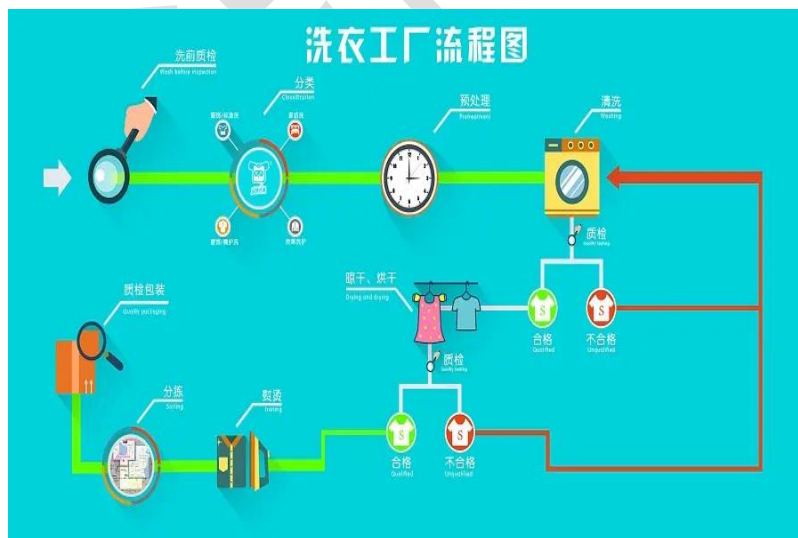
分支结构

- 根据条件，选择性地执行某段代码。
- 有 *if...else* 和 *switch-case* 两种分支语句。

循环结构

- 根据循环条件，重复性的执行某段代码。
- 有 *for*、*while*、*do-while* 三种循环语句。
- 补充：JDK5.0 提供了 *foreach* 循环，方便的遍历集合、数组元素。(第 12 章集合中讲解)

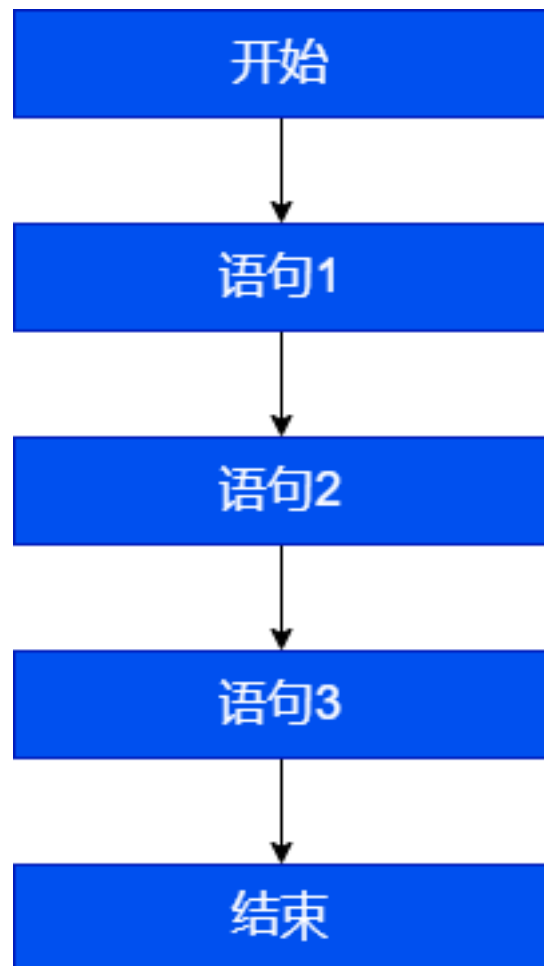
生活中、工业生产中流程控制举例



洗衣流程

1. 顺序结构

顺序结构就是程序从上到下逐行地执行。表达式语句都是顺序执行的。并且上一行对某个变量的修改对下一行会产生影响。



```
public class StatementTest{  
    public static void main(String[] args){  
        int x = 1;  
        int y = 2;  
        System.out.println("x = " + x);  
        System.out.println("y = " + y);  
        //对x、y 的值进行修改  
        x++;  
        y = 2 * x + y;  
        x = x * 10;  
    }  
}
```



```
        System.out.println("x = " + x);
        System.out.println("y = " + y);
    }
}
```

Java 中定义变量时采用合法的*前向引用*。如：

```
public static void main(String[] args) {
    int num1 = 12;
    int num2 = num1 + 2;
}
```

错误形式：

```
public static void main(String[] args) {
    int num2 = num1 + 2;
    int num1 = 12;
}
```

2. 分支语句

2.1 if-else 条件判断结构

2.1.1 基本语法

结构 1：单分支条件判断：if

格式：

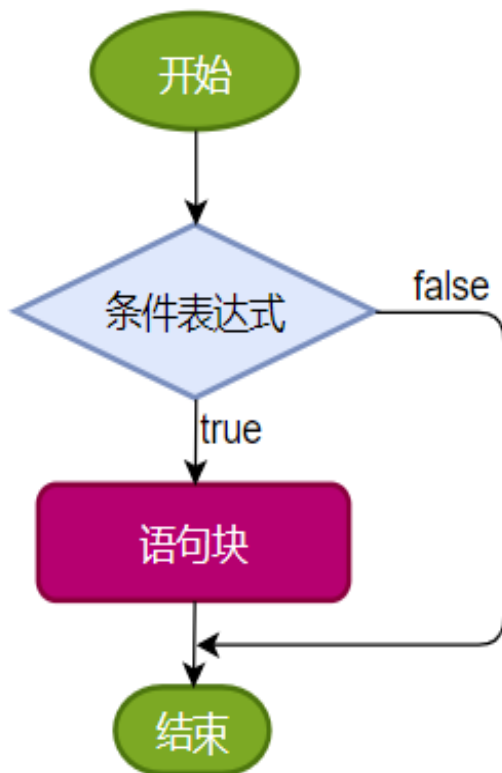
```
if(条件表达式) {
    语句块;
}
```

说明： 条件表达式必须是布尔表达式（关系表达式或逻辑表达式）或 布尔变量。

执行流程：

3. 首先判断条件表达式看其结果是 true 还是 false
4. 如果是 true 就执行语句块

5. 如果是 false 就不执行语句块



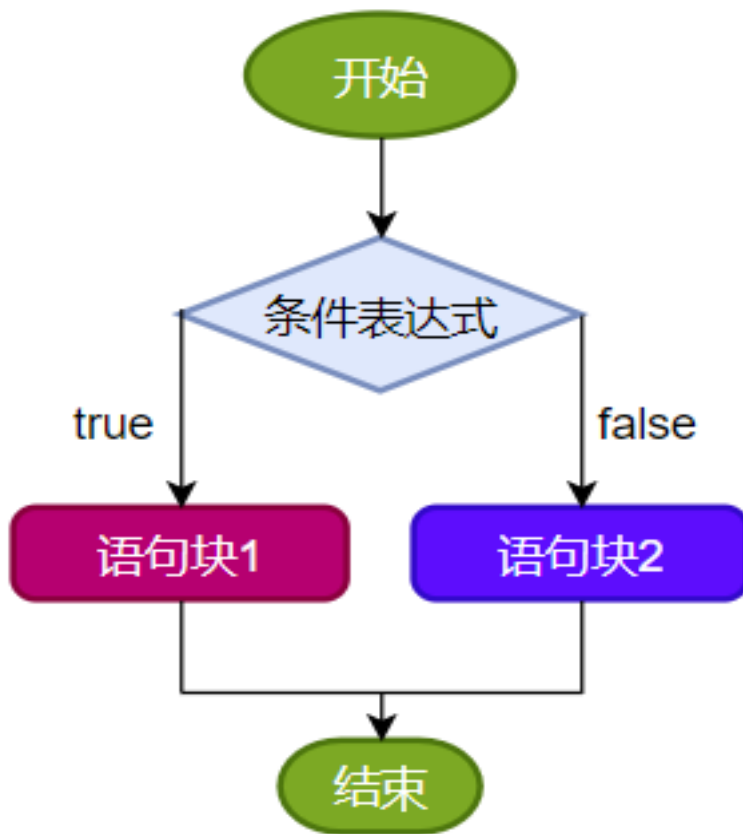
结构 2：双分支条件判断：if...else

格式：

```
if(条件表达式) {  
    语句块 1;  
}else {  
    语句块 2;  
}
```

执行流程：

6. 首先判断条件表达式看其结果是 true 还是 false
7. 如果是 true 就执行语句块 1
8. 如果是 false 就执行语句块 2



结构 3：多分支条件判断：if...else if...else

格式：

```
if (条件表达式 1) {  
    语句块 1;  
}  
else if (条件表达式 2) {  
    语句块 2;  
}  
...  
else if (条件表达式 n) {  
    语句块 n;  
}  
else {  
    语句块 n+1;  
}
```

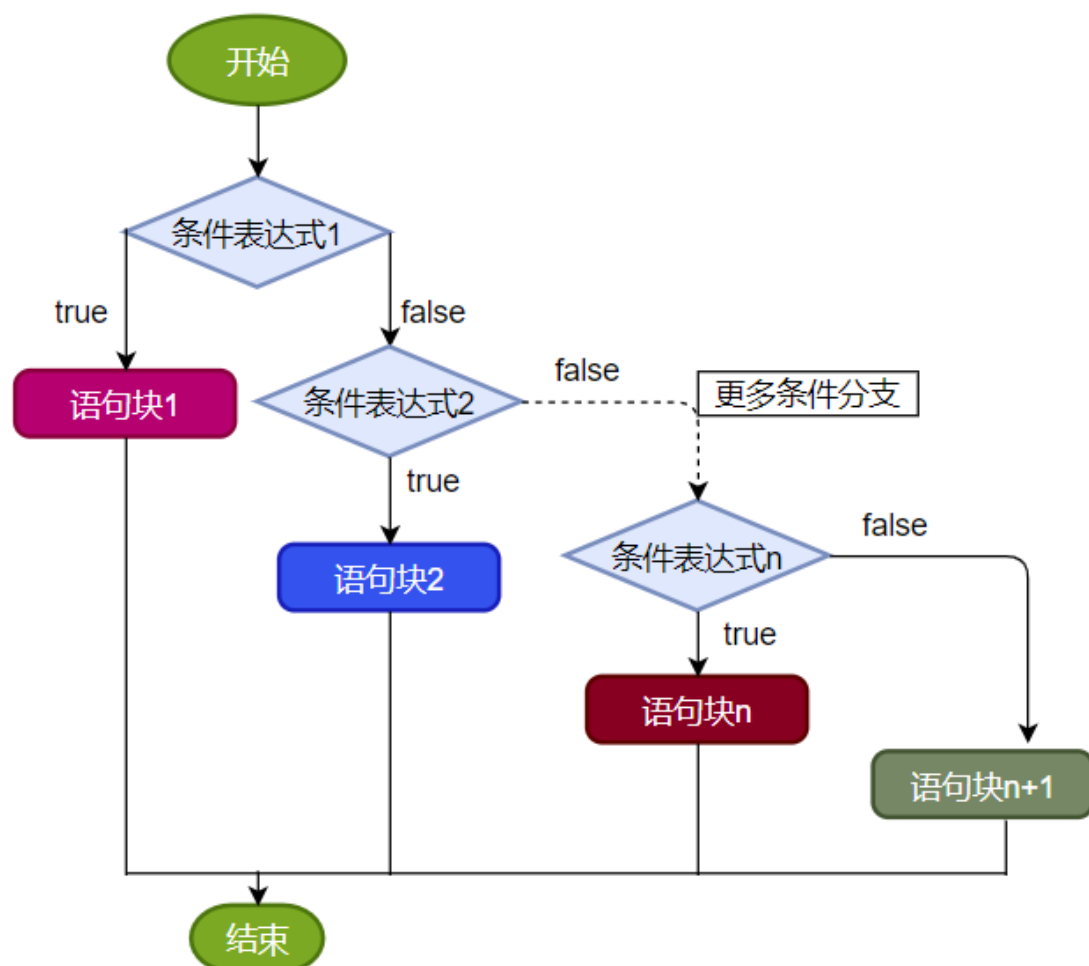
说明：一旦条件表达式为 true，则进入执行相应的语句块。执行完对应的语句块之后，就跳出当前结构。

执行流程:

9. 首先判断关系表达式 1 看其结果是 true 还是 false
10. 如果是 true 就执行语句块 1, 然后结束当前多分支
11. 如果是 false 就继续判断关系表达式 2 看其结果是 true 还是 false
12. 如果是 true 就执行语句块 2, 然后结束当前多分支
13. 如果是 false 就继续判断关系表达式...看其结果是 true 还是 false

...

n. 如果没有任何关系表达式为 true, 就执行语句块 n+1, 然后结束当前多分支。



2.1.2 应用举例

案例 1：成年人心率的正常范围是每分钟 60-100 次。体检时，如果心率不在此范围内，则提示需要做进一步的检查。

```
public class IfElseTest1 {  
    public static void main(String[] args){  
        int heartBeats = 89;  
  
        if(heartBeats < 60 || heartBeats > 100){  
            System.out.println("你需要做进一步的检查");  
        }  
  
        System.out.println("体检结束");  
    }  
}
```

案例 2：定义一个整数，判定是偶数还是奇数

```
public class IfElseTest2 {  
    public static void main(String[] args){  
        int a = 10;  
  
        if(a % 2 == 0) {  
            System.out.println(a + "是偶数");  
        } else{  
            System.out.println(a + "是奇数");  
        }  
    }  
}
```

案例 3：

岳小鹏参加 Java 考试，他和父亲岳不群达成承诺：

如果：

成绩为 100 分时，奖励一辆跑车；

成绩为(80, 99]时，奖励一辆山地自行车；

当成绩为[60,80]时，奖励环球影城一日游；

其它时，胖揍一顿。

说明：默认成绩是在[0,100]范围内

```

public class IfElseTest3 {
    public static void main(String[] args) {

        int score = 67; // 岳小鹏的期末成绩
        // 写法一：默认成绩范围为[0,100]
        if(score == 100){
            System.out.println("奖励一辆跑车");
        } else if(score > 80 && score <= 99){ // 错误的写法: }else if
(80 < score <= 99){
            System.out.println("奖励一辆山地自行车");
        } else if(score >= 60 && score <= 80){
            System.out.println("奖励环球影城玩一日游");
        }
        //else{
        // System.out.println("胖揍一顿");
        //}

        // 写法二:
        // 默认成绩范围为[0,100]
        if(score == 100){
            System.out.println("奖励一辆跑车");
        } else if(score > 80){
            System.out.println("奖励一辆山地自行车");
        } else if(score >= 60){
            System.out.println("奖励环球影城玩一日游");
        } else{
            System.out.println("胖揍一顿");
        }
    }
}

```

```

int score = 67; // 岳小鹏的期末成绩
// 写法一:
if(score == 100){
    System.out.println("奖励一辆跑车");
}else if(score > 80 && score <= 99){
    System.out.println("奖励一辆山地自行车");
}else if(score >= 60 && score <= 80){
    System.out.println("奖励环球影城玩一日游");
}
else{
    System.out.println("胖揍一顿");
}

```

条件之间没有交集，
各个条件顺序可以颠倒。

```

// 写法二:
// 默认成绩范围为[0,100]
if(score == 100){
    System.out.println("奖励一辆跑车");
}else if(score > 80){
    System.out.println("奖励一辆山地自行车");
}else if(score >= 60){
    System.out.println("奖励环球影城玩一日游");
}else{
    System.out.println("胖揍一顿");
}

```

条件之间存在交集，
则不能随意调换条件的顺序！

当条件表达式之间是“互斥”关系时（即彼此没有交集），条件判断语句及执行语句间顺序无所谓。

当条件表达式之间是“包含”关系时，“小上大下 / 子上父下”，否则范围小的条件表达式将不可能被执行。

2.1.3 if...else 嵌套

在 if 的语句块中，或者是在 else 语句块中，又包含了另外一个条件判断（可以是单分支、双分支、多分支），就构成了嵌套结构。

执行的特点：（1）如果是嵌套在 if 语句块中的，只有当外部的 if 条件满足，才会去判断内部的条件（2）如果是嵌套在 else 语句块中的，只有当外部的 if 条件不满足，进入 else 后，才会去判断内部的条件

案例 4：由键盘输入三个整数分别存入变量 num1、num2、num3，对它们进行排序(使用 if-else if-else)，并且从小到大输出。

```
class IfElseTest4 {
    public static void main(String[] args) {

        //声明num1,num2,num3 三个变量并赋值
        int num1 = 23,num2 = 32,num3 = 12;

        if(num1 >= num2){

            if(num3 >= num1)
                System.out.println(num2 + "-" + num1 + "-" + num3);
            else if(num3 <= num2)
                System.out.println(num3 + "-" + num2 + "-" + num1);
            else
                System.out.println(num2 + "-" + num3 + "-" + num1);
        }else{ //num1 < num2

            if(num3 >= num2){
                System.out.println(num1 + "-" + num2 + "-" + num3);
            }else if(num3 <= num1){
                System.out.println(num3 + "-" + num1 + "-" + num2);
            }else{
                System.out.println(num1 + "-" + num3 + "-" + num2);
            }
        }
    }
}
```

2.1.4 其它说明

- 语句块只有一条执行语句时，一对{}可以省略，但建议保留
- 当 if-else 结构是“多选一”时，最后的 else 是可选的，根据需要可以省略

2.1.5 练习

练习 1:

//1) 对下列代码，若有输出，指出输出结果。

```
int x = 4;
int y = 1;
if (x > 2) {
    if (y > 2)
        System.out.println(x + y);
        System.out.println("atguigu");
} else
    System.out.println("x is " + x);
```

练习 2:

```
boolean b = true;
//如果写成 if(b=false) 能编译通过吗？如果能，结果是？
if(b == false)    //建议: if(!b)
    System.out.println("a");
else if(b)
    System.out.println("b");
else if(!b)
    System.out.println("c");
else
    System.out.println("d");
```

练习 3:

定义两个整数，分别为 small 和 big，如果第一个整数 small 大于第二个整数 big，就交换。输出显示 small 和 big 变量的值。

```
public class IfElseExer3 {
    public static void main(String[] args) {
        int small = 10;
        int big = 9;

        if (small > big) {
            int temp = small;
            small = big;
        }
    }
}
```

```

        big = temp;
    }
    System.out.println("small=" + small + ",big=" + big);
}
}

```

练习 4: 小明参加期末 Java 考试，通过考试成绩，判断其 Java 等级，成绩范围 [0,100]

- 90-100 优秀
- 80-89 好
- 70-79 良
- 60-69 及格
- 60 以下 不及格

```

import java.util.Scanner;
//写法一:
public class IfElseExer4 {
    public static void main(String[] args) {
        System.out.print("小明的期末 Java 成绩是: [0,100]");
        int score = 89;

        if (score < 0 || score > 100) {
            System.out.println("你的成绩是错误的");
        } else if (score >= 90 && score <= 100) {
            System.out.println("你的成绩属于优秀");
        } else if (score >= 80 && score < 90) {
            System.out.println("你的成绩属于好");
        } else if (score >= 70 && score < 80) {
            System.out.println("你的成绩属于良");
        } else if (score >= 60 && score < 70) {
            System.out.println("你的成绩属于及格");
        } else {
            System.out.println("你的成绩属于不及格");
        }
    }
}

```

```

import java.util.Scanner;
//写法二:
public class IfElseExer4 {

```

```

public static void main(String[] args) {
    System.out.print("小明的期末 Java 成绩是: [0,100]");
    int score = 89;

    if (score < 0 || score > 100) {
        System.out.println("你的成绩是错误的");
    } else if (score >= 90) {
        System.out.println("你的成绩属于优秀");
    } else if (score >= 80) {
        System.out.println("你的成绩属于好");
    } else if (score >= 70) {
        System.out.println("你的成绩属于良");
    } else if (score >= 60) {
        System.out.println("你的成绩属于及格");
    } else {
        System.out.println("你的成绩属于不及格");
    }
}
}

```

练习 5:

编写程序，声明 2 个 int 型变量并赋值。判断两数之和，如果大于等于 50，打印“hello world!”

```

public class IfElseExer5 {

    public static void main(String[] args) {
        int num1 = 12, num2 = 32;

        if (num1 + num2 >= 50) {
            System.out.println("hello world!");
        }
    }
}

```

练习 6:

编写程序，声明 2 个 double 型变量并赋值。判断第一个数大于 10.0，且第 2 个数小于 20.0，打印两数之和。否则，打印两数的乘积。

```

public class IfElseExer6 {

    public static void main(String[] args) {

```

```

double d1 = 21.2,d2 = 12.3;

if(d1 > 10.0 && d2 < 20.0){
    System.out.println("两数之和为: " + (d1 + d2));
}else{
    System.out.println("两数乘积为: " + (d1 * d2));
}
}
}

```

练习 7：判断水的温度

如果大于 95℃，则打印“开水”；

如果大于 70℃且小于等于 95℃，则打印“热水”；

如果大于 40℃且小于等于 70℃，则打印“温水”；

如果小于等于 40℃，则打印“凉水”。

```

public class IfElseExer7 {

    public static void main(String[] args) {
        int waterTemperature = 85;

        if(waterTemperature > 95){
            System.out.println("开水");
        }else if(waterTemperature > 70 && waterTemperature <= 95){
            System.out.println("热水");
        }else if(waterTemperature > 40 && waterTemperature <= 70){
            System.out.println("温水");
        }else{
            System.out.println("凉水");
        }
    }
}
}

```

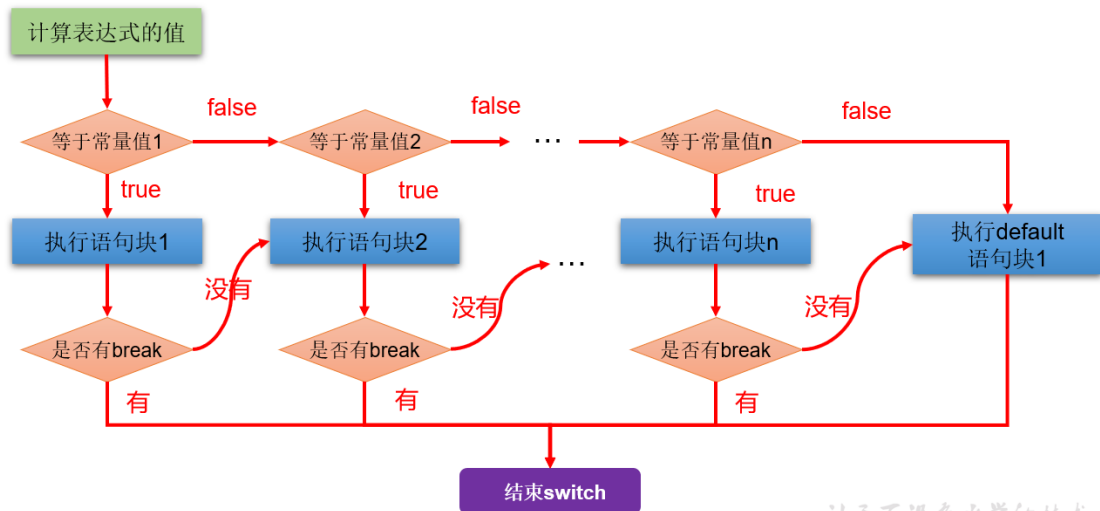
2.2 switch-case 选择结构

2.2.1 基本语法

语法格式：

```
switch(表达式){  
    case 常量值 1:  
        语句块 1;  
        //break;  
    case 常量值 2:  
        语句块 2;  
        //break;  
    // ...  
    [default:  
        语句块 n+1;  
        break;  
    ]  
}
```

执行流程图：



让天下没有难学的技术

执行过程：

第 1 步：根据 switch 中表达式的值，依次匹配各个 case。如果表达式的值等于某个 case 中的常量值，则执行对应 case 中的执行语句。

第 2 步：执行完此 case 的执行语句以后，
情况 1：如果遇到 break,则执行 break 并跳出当前的 switch-case 结构
情况 2：如果没有遇到 break，则会继续执行当前 case 之后的其它 case 中的执行语句。--->case 穿透 ... 直到遇到 break 关键字或执行完所有的 case 及 default 的执行语句，跳出当前的 switch-case 结构

使用注意点：

- switch(表达式)中表达式的值必须是下述几种类型之一：byte, short, char, int, 枚举 (jdk 5.0), String (jdk 7.0);
- case 子句中的值必须是常量，不能是变量名或不确定的表达式值或范围；
- 同一个 switch 语句，所有 case 子句中的常量值互不相同；
- break 语句用来在执行完一个 case 分支后使程序跳出 switch 语句块；
如果没有 break，程序会顺序执行到 switch 结尾；
- default 子句是可选的。同时，位置也是灵活的。当没有匹配的 case 时，执行 default 语句。

2.2.2 应用举例

案例 1：

```
public class SwitchCaseTest1 {  
    public static void main(String args[]) {  
        int num = 1;  
        switch(num){  
            case 0:  
                System.out.println("zero");  
                break;  
            case 1:  
                System.out.println("one");  
        }  
    }  
}
```

```

        break;
    case 2:
        System.out.println("two");
        break;
    case 3:
        System.out.println("three");
        break;
    default:
        System.out.println("other");
        //break;
    }
}
}

```

案例 2:

```

public class SwitchCaseTest2 {
    public static void main(String args[]) {
        String season = "summer";
        switch (season) {
            case "spring":
                System.out.println("春暖花开");
                break;
            case "summer":
                System.out.println("夏日炎炎");
                break;
            case "autumn":
                System.out.println("秋高气爽");
                break;
            case "winter":
                System.out.println("冬雪皑皑");
                break;
            default:
                System.out.println("季节输入有误");
                break;
        }
    }
}

```

错误举例:

```

int key = 10;
switch(key){
    case key > 0 :

```

```

        System.out.println("正数");
        break;
    case key < 0:
        System.out.println("负数");
        break;
    default:
        System.out.println("零");
        break;
}

```

案例 3：使用 switch-case 实现：对学生成绩大于 60 分的，输出“合格”。低于 60 分的，输出“不合格”。

```

class SwitchCaseTest3 {
    public static void main(String[] args) {

        int score = 67;
        /*
        写法 1：极不推荐
        switch(score){
            case 0:
                System.out.println("不及格");
                break;
            case 1:
                System.out.println("不及格");
                break;
            //...

            case 60:
                System.out.println("及格");
                break;
            //...略...

        }
        */

        //写法 2:
        switch(score / 10){
            case 0:
            case 1:
            case 2:
            case 3:
            case 4:

```



```

        case 5:
            System.out.println("不及格");
            break;
        case 6:
        case 7:
        case 8:
        case 9:
        case 10:
            System.out.println("及格");
            break;
        default:
            System.out.println("输入的成绩有误");
            break;
    }

    // 写法3:
    switch(score / 60){
        case 0:
            System.out.println("不及格");
            break;
        case 1:
            System.out.println("及格");
            break;
        default:
            System.out.println("输入的成绩有误");
            break;
    }
}
}

```

2.2.3 利用 case 的穿透性

在 switch 语句中，如果 case 的后面不写 break，将出现穿透现象，也就是一旦匹配成功，不会在判断下一个 case 的值，直接向后运行，直到遇到 break 或者整个 switch 语句结束，执行终止。

案例 4：编写程序：从键盘上输入 2023 年的“month”和“day”，要求通过程序输出输入的日期为 2023 年的第几天。

```
import java.util.Scanner;
```

```
class SwitchCaseTest4 {  
    public static void main(String[] args) {
```

```
        Scanner scan = new Scanner(System.in);
```

```
        System.out.println("请输入 2023 年的 month:");
```

```
        int month = scan.nextInt();
```

```
        System.out.println("请输入 2023 年的 day:");
```

```
        int day = scan.nextInt();
```

//这里就不针对month 和 day 进行合法性的判断了, 以后可以使用正则表达式进行校验。

```
        int sumDays = 0; //记录总天数
```

```
//写法1 : 不推荐 (存在冗余的数据)
```

```
/*
```

```
    switch(month){
```

```
        case 1:
```

```
            sumDays = day;
```

```
            break;
```

```
        case 2:
```

```
            sumDays = 31 + day;
```

```
            break;
```

```
        case 3:
```

```
            sumDays = 31 + 28 + day;
```

```
            break;
```

```
//....
```

```
        case 12:
```

```
//sumDays = 31 + 28 + ... + 30 + day;
```

```
            break;
```

```
    }
```

```
*/
```

```
//写法2: 推荐
```

```
    switch(month){
```

```
        case 12:
```

```
            sumDays += 30; //这个30 是代表11 月份的满月天数
```

```
        case 11:
```

```

        sumDays += 31; // 这个 31 是代表 10 月份的满月天数
    case 10:
        sumDays += 30; // 这个 30 是代表 9 月份的满月天数
    case 9:
        sumDays += 31; // 这个 31 是代表 8 月份的满月天数
    case 8:
        sumDays += 31; // 这个 31 是代表 7 月份的满月天数
    case 7:
        sumDays += 30; // 这个 30 是代表 6 月份的满月天数
    case 6:
        sumDays += 31; // 这个 31 是代表 5 月份的满月天数
    case 5:
        sumDays += 30; // 这个 30 是代表 4 月份的满月天数
    case 4:
        sumDays += 31; // 这个 31 是代表 3 月份的满月天数
    case 3:
        sumDays += 28; // 这个 28 是代表 2 月份的满月天数
    case 2:
        sumDays += 31; // 这个 31 是代表 1 月份的满月天数
    case 1:
        sumDays += day; // 这个 day 是代表当月的第几天
    }

    System.out.println(month + "月" + day + "日是 2023 年的第" + sum
Days + "天");
    // 关闭资源
    scan.close();
}
}

```

拓展：

从键盘分别输入年、月、日，判断这一天是当年的第几天

注：判断一年是否是闰年的标准：

- 1) 可以被 4 整除，但不可被 100 整除
或
- 2) 可以被 400 整除

例如：1900，2200 等能被 4 整除，但同时能被 100 整除，但不能被 400 整除，不是闰年

```
import java.util.Scanner;
```

```

public class SwitchCaseTest04 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.print("请输入 year:");
        int year = scanner.nextInt();

        System.out.print("请输入 month:");
        int month = scanner.nextInt();

        System.out.print("请输入 day:");
        int day = scanner.nextInt();

        //判断这一天是当年的第几天==>从1月1日开始, 累加到xx月xx日这一
        天

        //(1)[1,month-1]个月满月天数
        //(2)单独考虑2月份是否是29天(依据是看year是否是闰年)
        //(3)第month个月的day天

        //声明一个变量 days, 用来存储总天数
        int sumDays = 0;

        //累加[1,month-1]个月满月天数
        switch (month) {
            case 12:
                //累加的1-11月
                sumDays += 30; //这个30是代表11月份的满月天数
                //这里没有break, 继续往下走
            case 11:
                //累加的1-10月
                sumDays += 31; //这个31是代表10月的满月天数
                //这里没有break, 继续往下走
            case 10:
                sumDays += 30; //9月
            case 9:
                sumDays += 31; //8月
            case 8:
                sumDays += 31; //7月
            case 7:
                sumDays += 30; //6月
            case 6:
                sumDays += 31; //5月
            case 5:

```

```

        sumDays += 30; //4 月
    case 4:
        sumDays += 31; //3 月
    case 3:
        sumDays += 28; //2 月
        //在这里考虑是否可能是 29 天
        if (year % 4 == 0 && year % 100 != 0 || year % 400 ==
0) {
            sumDays++; //多加 1 天
        }
    case 2:
        sumDays += 31; //1 月
    case 1:
        sumDays += day; //第 month 月的 day 天
    }

    //输出结果
    System.out.println(year + "年" + month + "月" + day + "日是这一
年的第" + sumDays + "天");

    scanner.close();
}
}

```

案例 5：根据指定的月份输出对应季节

```

import java.util.Scanner;

/*
 * 需求：指定一个月份，输出该月份对应的季节。一年有四季：
 *      3,4,5  春季
 *      6,7,8  夏季
 *      9,10,11 秋季
 *      12,1,2 冬季
 */
public class SwitchCaseTest5 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("请输入月份：");
        int month = input.nextInt();

        /*
        switch(month) {
            case 1:

```

```
        System.out.println("冬季");
        break;
    case 2:
        System.out.println("冬季");
        break;
    case 3:
        System.out.println("春季");
        break;
    case 4:
        System.out.println("春季");
        break;
    case 5:
        System.out.println("春季");
        break;
    case 6:
        System.out.println("夏季");
        break;
    case 7:
        System.out.println("夏季");
        break;
    case 8:
        System.out.println("夏季");
        break;
    case 9:
        System.out.println("秋季");
        break;
    case 10:
        System.out.println("秋季");
        break;
    case 11:
        System.out.println("秋季");
        break;
    case 12:
        System.out.println("冬季");
        break;
    default:
        System.out.println("你输入的月份有误");
        break;
}
*/
```

// 改进版

```
switch(month) {
    case 1:
```

```

        case 2:
        case 12:
            System.out.println("冬季");
            break;
        case 3:
        case 4:
        case 5:
            System.out.println("春季");
            break;
        case 6:
        case 7:
        case 8:
            System.out.println("夏季");
            break;
        case 9:
        case 10:
        case 11:
            System.out.println("秋季");
            break;
        default:
            System.out.println("你输入的月份有误");
            break;
    }

    input.close();
}
}

```

常见错误实现:

```

switch(month){
    case 3|4|5://3|4|5 用了位运算符, 11 | 100 | 101 结果是 111 是 7
        System.out.println("春季");
        break;
    case 6|7|8://6|7|8 用了位运算符, 110 | 111 | 1000 结果是 1111 是 15
        System.out.println("夏季");
        break;
    case 9|10|11://9|10|11 用了位运算符, 1001 | 1010 | 1011 结果是 1011
        System.out.println("秋季");
        break;
    case 12|1|2://12|1|2 用了位运算符, 1100 | 1 | 10 结果是 1111, 是 15
        System.out.println("冬季");
        break;
}

```

```
    default:
        System.out.println("输入有误");
}
```

使用 if-else 实现：

```
if ((month == 1) || (month == 2) || (month == 12)) {
    System.out.println("冬季");
} else if ((month == 3) || (month == 4) || (month == 5)) {
    System.out.println("春季");
} else if ((month == 6) || (month == 7) || (month == 8)) {
    System.out.println("夏季");
} else if ((month == 9) || (month == 10) || (month == 11)) {
    System.out.println("秋季");
} else {
    System.out.println("你输入的月份有误");
}
```

2.2.4 if-else 语句与 switch-case 语句比较

- 结论：凡是使用 switch-case 的结构都可以转换为 if-else 结构。反之，不成立。
- 开发经验：如果既可以使用 switch-case，又可以使用 if-else，建议使用 switch-case。因为效率稍高。
- 细节对比：
 - if-else 语句优势
 - if 语句的条件是一个布尔类型值，if 条件表达式为 true 则进入分支，可以用于范围的判断，也可以用于等值的判断，*使用范围更广*。
 - switch 语句的条件是一个常量值 (byte,short,int,char,枚举,String)，只能判断某个变量或表达式的结果是否等于某个常量值，*使用场景较狭窄*。
 - switch 语句优势
 - 当条件是判断某个变量或表达式是否等于某个固定的常量值时，使用 if 和 switch 都可以，习惯上使用 switch 更多。因为*效率稍高*。当条件是区间范围的判断时，只能使用 if 语句。
 - 使用 switch 可以利用*穿透性*，同时执行多个分支，而 if...else 没有穿透性。

- 案例：只能使用 if-else

从键盘输入一个整数，判断是正数、负数、还是零。

```
import java.util.Scanner;

public class IfOrSwitchDemo {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("请输入整数: ");
        int num = input.nextInt();

        if (num > 0) {
            System.out.println(num + "是正整数");
        } else if (num < 0) {
            System.out.println(num + "是负整数");
        } else {
            System.out.println(num + "是零");
        }

        input.close();
    }
}
```

2.2.5 练习

练习 1：从键盘输入星期的整数值，输出星期的英文单词

```
import java.util.Scanner;

public class SwitchCaseExer1 {
    public static void main(String[] args) {
        //定义指定的星期
        Scanner input = new Scanner(System.in);
        System.out.print("请输入星期值: ");
        int weekday = input.nextInt();

        //switch 语句实现选择
        switch(weekday) {
            case 1:
                System.out.println("Monday");
                break;
            case 2:
```

```

        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
    case 4:
        System.out.println("Thursday");
        break;
    case 5:
        System.out.println("Friday");
        break;
    case 6:
        System.out.println("Saturday");
        break;
    case 7:
        System.out.println("Sunday");
        break;
    default:
        System.out.println("你输入的星期值有误! ");
        break;
    }
}

input.close();
}
}

```

练习 2:

使用 **switch** 把小写类型的 `char` 型转为大写。只转换 `a`, `b`, `c`, `d`, `e`。其它的输出“other”。

```

public class SwitchCaseExer2 {

    public static void main(String[] args) {

        char word = 'c';
        switch (word) {
            case 'a':
                System.out.println("A");
                break;
            case 'b':
                System.out.println("B");
                break;
            case 'c':

```

```

        System.out.println("C");
        break;
    case 'd':
        System.out.println("D");
        break;
    case 'e':
        System.out.println("E");
        break;
    default :
        System.out.println("other");
    }
}
}

```

练习 3:

编写程序：从键盘上读入一个学生成绩，存放在变量 `score` 中，根据 `score` 的值输出其对应的成绩等级：

<code>score >= 90</code>	等级： A
<code>70 <= score < 90</code>	等级： B
<code>60 <= score < 70</code>	等级： C
<code>score < 60</code>	等级： D

方式一：使用 if-else

方式二：使用 switch-case: `score / 10: 0 - 10`

```

public class SwitchCaseExer3 {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);
        System.out.println("请输入学生成绩: ");
        int score = scan.nextInt();

        char grade; // 记录学生等级
        // 方式1:
        //      if(score >= 90){
        //          grade = 'A';
        //      }else if(score >= 70 && score < 90){
        //          grade = 'B';
        //      }else if(score >= 60 && score < 70){
        //          grade = 'C';
        //      }else{

```

```

//          grade = 'D';
//      }

//方式2:
switch(score / 10){
    case 10:
    case 9:
        grade = 'A';
        break;
    case 8:
    case 7:
        grade = 'B';
        break;
    case 6:
        grade = 'C';
        break;
    default :
        grade = 'D';
}

System.out.println("学生成绩为" + score + ",对应的等级为" + grade);

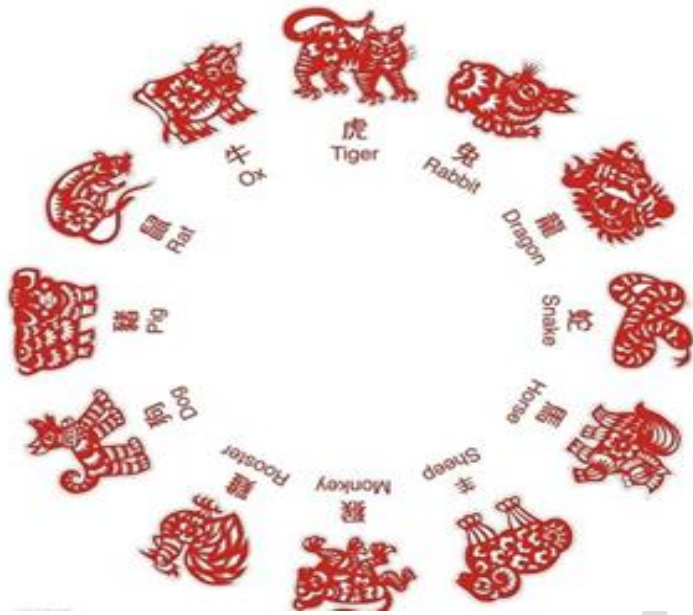
scan.close();
}
}

```

练习 4:

编写一个程序，为一个给定的年份找出其对应的中国生肖。中国的生肖基于 12 年一个周期，每年用一个动物代表：rat、ox、tiger、rabbit、dragon、snake、horse、sheep、monkey、rooster、dog、pig。

提示：2022 年：虎 $2022 \% 12 == 6$



```
public class SwitchCaseExer4 {  
    public static void main(String[] args){  
        //从键盘输入一个年份  
        Scanner input = new Scanner(System.in);  
        System.out.print("请输入年份: ");  
        int year = input.nextInt();  
        input.close();  
  
        //判断  
        switch(year % 12){  
            case 0:  
                System.out.println(year + "是猴年");  
                break;  
            case 1:  
                System.out.println(year + "是鸡年");  
                break;  
            case 2:  
                System.out.println(year + "是狗年");  
                break;  
            case 3:  
                System.out.println(year + "是猪年");  
                break;  
            case 4:  
                System.out.println(year + "是鼠年");  
                break;  
            case 5:  
                System.out.println(year + "是牛年");  
                break;  
            case 6:  
                System.out.println(year + "是虎年");  
                break;  
            case 7:  
                System.out.println(year + "是兔年");  
                break;  
            case 8:  
                System.out.println(year + "是龙年");  
                break;  
            case 9:  
                System.out.println(year + "是蛇年");  
                break;  
            case 10:  
                System.out.println(year + "是马年");  
                break;  
            case 11:  
                System.out.println(year + "是羊年");  
                break;  
        }  
    }  
}
```

```

        break;
    case 6:
        System.out.println(year + "是虎年");
        break;
    case 7:
        System.out.println(year + "是兔年");
        break;
    case 8:
        System.out.println(year + "是龙年");
        break;
    case 9:
        System.out.println(year + "是蛇年");
        break;
    case 10:
        System.out.println(year + "是马年");
        break;
    case 11:
        System.out.println(year + "是羊年");
        break;
    default:
        System.out.println(year + "输入错误");
    }
}
}

```

练习 5：押宝游戏

随机产生 3 个 1-6 的整数，如果三个数相等，那么称为“豹子”，如果三个数之和大于 9，称为“大”，如果三个数之和小于等于 9，称为“小”，用户从键盘输入押的是“豹子”、“大”、“小”，并判断是否猜对了

提示：随机数 `Math.random()` 产生 `[0,1)` 范围内的小数

如何获取 `[a,b]` 范围内的随机整数呢？`(int)(Math.random() * (b - a + 1)) + a`

```
C:\Users\songhk\Desktop>java Exercise
请押宝（豹子、大、小）：豹子
a, b, c分别是：3, 5, 3
猜错了

C:\Users\songhk\Desktop>java Exercise
请押宝（豹子、大、小）：大
a, b, c分别是：5, 6, 4
猜中了
```

```
import java.util.Scanner;
```

```
public class SwitchCaseExer5 {
    public static void main(String[] args) {
        //1、随机产生3个1-6的整数
        int a = (int)(Math.random()*6 + 1);
        int b = (int)(Math.random()*6 + 1);
        int c = (int)(Math.random()*6 + 1);

        //2、押宝
        Scanner input = new Scanner(System.in);
        System.out.print("请押宝（豹子、大、小）：");
        String ya = input.next();
        input.close();

        //3、判断结果
        boolean result = false;
        //switch支持String类型
        switch (ya){
            case "豹子": result = a == b && b == c; break;
            case "大": result = a + b + c > 9; break;
            case "小": result = a + b + c <= 9; break;
            default: System.out.println("输入有误！");
        }

        System.out.println("a,b,c 分别是： " + a + ", " + b + ", " + c );
        System.out.println(result ? "猜中了" : "猜错了");
    }
}
```

练习 6:

使用 switch 语句改写下列 if 语句:

```
int a = 3;
```

```
int x = 100;
if(a==1)
    x+=5;
else if(a==2)
    x+=10;
else if(a==3)
    x+=16;
else
    x+=34;

int a = 3;
int x = 100;

switch(a){
    case 1:
        x += 5;
        break;
    case 2:
        x += 10;
        break;
    case 3:
        x += 16;
        break;
    default :
        x += 34;
}
```

3. 循环语句

- 理解：循环语句具有在*某些条件*满足的情况下，*反复执行*特定代码的功能。
- 循环结构分类：
 - for 循环
 - while 循环
 - do-while 循环
- 循环结构*四要素*：
 - 初始化部分
 - 循环条件部分
 - 循环体部分

– 迭代部分

3.1 for 循环

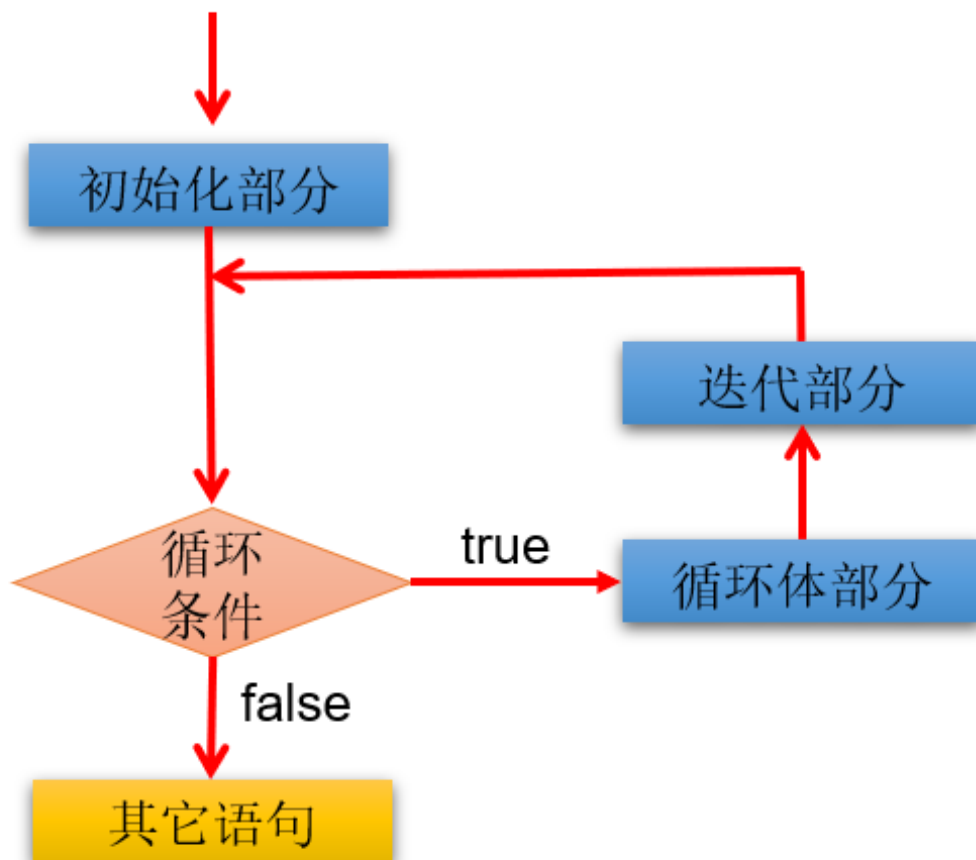
3.1.1 基本语法

语法格式：

```
for (①初始化部分; ②循环条件部分; ④迭代部分) {  
    ③循环体部分;  
}
```

执行过程：①-②-③-④-②-③-④-②-③-④-.....-②

图示：



说明：

- for(;;)中的两个; 不能多也不能少
- ①初始化部分可以声明多个变量，但必须是同一个类型，用逗号分隔
- ②循环条件部分为 boolean 类型表达式，当值为 false 时，退出循环
- ④可以有多个变量更新，用逗号分隔

3.1.2 应用举例

案例 1：使用 for 循环重复执行某些语句

题目：输出 5 行 HelloWorld

```
public class ForTest1 {  
    public static void main(String[] args) {  
        //需求1：控制台输出5行Hello World!  
        //写法1：  
        //System.out.println("Hello World!");  
        //System.out.println("Hello World!");  
        //System.out.println("Hello World!");  
        //System.out.println("Hello World!");  
        //System.out.println("Hello World!");  
  
        //写法2：  
        for(int i = 1; i <= 5; i++){  
            System.out.println("Hello World!");  
        }  
    }  
}
```

案例 2：格式的多样性

题目：写出输出的结果

```
public class ForTest2 {  
    public static void main(String[] args) {  
        int num = 1;  
        for(System.out.print("a"); num < 3; System.out.print("c"), num+  
        +){
```

```

        System.out.print("b");
    }
}
}

```

案例 3：累加的思想

题目：遍历 1-100 以内的偶数，并获取偶数的个数，获取所有的偶数的和

```

public class ForTest3 {
    public static void main(String[] args) {
        int count = 0; // 记录偶数的个数
        int sum = 0; // 记录偶数的和

        for(int i = 1; i <= 100; i++){

            if(i % 2 == 0){
                System.out.println(i);
                count++;
                sum += i;
            }

            //System.out.println("偶数的个数为: " + count);
        }

        System.out.println("偶数的个数为: " + count);
        System.out.println("偶数的总和为: " + sum);
    }
}

```

案例 4：结合分支结构使用

题目：输出所有的水仙花数，所谓水仙花数是指一个 3 位数，其各个位上数字

立方和等于其本身。例如： $153 = 1*1*1 + 3*3*3 + 5*5*5$

```

public class ForTest4 {
    public static void main(String[] args) {
        // 定义统计变量，初始化值是 0
        int count = 0;

        // 获取三位数，用 for 循环实现
    }
}

```

```

for(int x = 100; x < 1000; x++) {
    //获取三位数的个位, 十位, 百位
    int ge = x % 10;
    int shi = x / 10 % 10;
    int bai = x / 100;

    //判断这个三位数是否是水仙花数, 如果是, 统计变量++
    if((ge*ge*ge+shi*shi*shi+bai*bai*bai) == x) {
        System.out.println("水仙花数: " + x);
        count++;
    }
}

//输出统计结果就可以了
System.out.println("水仙花数共有"+count+"个");
}
}

```

拓展:

打印出四位数字中“个位+百位”等于“十位+千位”并且个位数为偶数, 千位数为奇数的数字, 并打印符合条件的数字的个数。

案例 5: 结合 break 的使用

说明: 输入两个正整数 m 和 n, 求其最大公约数和最小公倍数。

比如: 12 和 20 的最大公约数是 4, 最小公倍数是 60。

```

public class ForTest5 {
    public static void main(String[] args) {
        //需求 1: 最大公约数
        int m = 12, n = 20;
        //取出两个数中的较小值
        int min = (m < n) ? m : n;

        for (int i = min; i >= 1; i--) {
            //for(int i = 1; i <= min; i++){

                if (m % i == 0 && n % i == 0) {
                    System.out.println("最大公约数是: " + i); //公约数
                }
            }
        }
    }
}

```

```

        break; //跳出当前循环结构
    }
}

//需求2: 最小公倍数
//取出两个数中的较大值
int max = (m > n) ? m : n;

for (int i = max; i <= m * n; i++) {

    if (i % m == 0 && i % n == 0) {

        System.out.println("最小公倍数是: " + i); //公倍数

        break;
    }
}
}
}

```

说明:

1、我们可以在循环中使用 break。一旦执行 break，就跳出当前循环结构。

2、小结：如何结束一个循环结构？

结束情况 1：循环结构中的循环条件部分返回 false

结束情况 2：循环结构中执行了 break。

3、如果一个循环结构不能结束，那就是一个死循环！我们开发中要避免出现死循环。

3.1.3 练习

练习 1：打印 1~100 之间所有奇数的和

```
public class ForExer1 {  
  
    public static void main(String[] args) {  
  
        int sum = 0; // 记录奇数的和  
        for (int i = 1; i < 100; i++) {  
            if (i % 2 != 0) {  
                sum += i;  
            }  
        }  
        System.out.println("奇数总和为: " + sum);  
    }  
}
```

练习 2：打印 1~100 之间所有是 7 的倍数的整数的个数及总和（体会设置计数器的思想）

```
public class ForExer2 {  
  
    public static void main(String[] args) {  
  
        int sum = 0; // 记录总和  
        int count = 0; // 记录个数  
        for (int i = 1; i < 100; i++) {  
            if (i % 7 == 0) {  
                sum += i;  
                count++;  
            }  
        }  
        System.out.println("1~100 之间所有是 7 的倍数的整数的和为: " + sum);  
        System.out.println("1~100 之间所有是 7 的倍数的整数的个数为: " + count);  
    }  
}
```